PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2003-131772
(43)Date of publication of application : 09.05.2003

(51)Int.Cl. G06F 3/00
G06F 3/16
G10L 15/00
G10L 15/28

(21)Application number : 2002-131950
(22)Date of filing : 07.05.2002

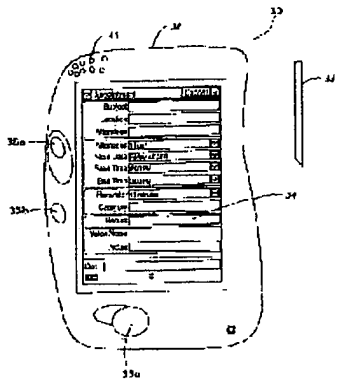
(71)Applicant : MICROSOFT CORP
(72)Inventor : WANG KUANSAN
HON HSIAO-WUEN

(30)Priority

Priority number	Priority date	Priority country
2001 289041	04.05.2001	US
2001 960233	20.09.2001	US
2002 117141	05.04.2002	US

(54) MARKUP LANGUAGE EXTENSIONS FOR RECOGNITION USABLE IN WEB

(57)Abstract:
PROBLEM TO BE SOLVED: To improve a method for providing speech recognition and so on through a client/server architecture such as the Internet and so on.
SOLUTION: Documents of a markup language executed on a client device in a client/server system includes instructions indicating a grammar for association with input entered through the client device.



LEGAL STATUS

[Date of request for examination] 06.05.2005
[Date of sending the examiner's decision of rejection]
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]

(原文除く 51 頁)

(19)日本国特許庁（JP） (12) 公開特許公報（A） (11)特許出願公開番号
特開2003－131772
(P2003－131772A)
(43)公開日 平成15年5月9日(2003.5.9)

(51)Int.Cl.'		識別記号	FI		テーマコード'(参考)
G06F	3/00	601	G06F	3/00	601 5D015
	3/16	320		3/16	320H 5E501
G10L	15/00		G10L	3/00	551P
	15/28				551A

審査請求 未請求 請求項の数25 OL 外国語出願 (全166頁)

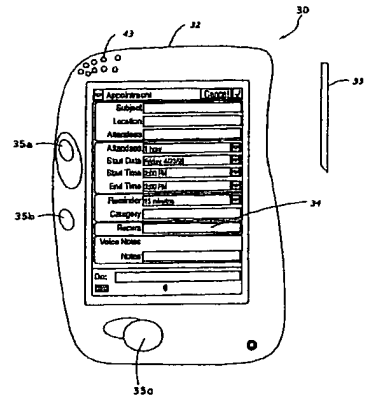
(21)出願番号 特願2002－131950(P2002－131950)
(22)出願日 平成14年5月7日(2002.5.7)
(31)優先権主張番号 60／289，041
(32)優先日 平成13年5月4日(2001.5.4)
(33)優先権主張国 米国（US）
(31)優先権主張番号 09／960，233
(32)優先日 平成13年9月20日(2001.9.20)
(33)優先権主張国 米国（US）
(31)優先権主張番号 10／117，141
(32)優先日 平成14年4月5日(2002.4.5)
(33)優先権主張国 米国（US）

(71)出願人 391055933
マイクロソフト コーポレイション
MICROSOFT CORPORATION
アメリカ合衆国 ワシントン州 98052－6399 レッドモンド ワン マイクロソフト ウェイ （番地なし）
(74)代理人 100077481
弁理士 谷 義一 （外2名）

最終頁に続く

(54)【発明の名称】 Webで使用可能な認識のためのマークアップ言語拡張部

(57)【要約】
【課題】 インターネットなどのサーバ／クライアントアーキテクチャで音声認識等を提供するための方法を改善する。
【解決手段】 クライアント／サーバシステム中のクライアントデバイスで実行するためのマークアップ言語の文書が、クライアントデバイスを介して入力された入力と関連付けるための文法を示す命令を含む。



1

【特許請求の範囲】

【請求項１】　クライアント／サーバシステム中のクライアントデバイスで実行するためのマークアップ言語を有するコンピュータ読取り可能媒体であって、前記マークアップ言語が、クライアントデバイスを介して入力された入力と関連付けるための文法を示す命令を含むことを特徴とするコンピュータ読取り可能媒体。

【請求項２】　文法を示す前記命令が、前記文法の場所の参照を提供することを特徴とする請求項１に記載のコンピュータ読取り可能媒体。

【請求項３】　前記文法が、音声認識、手書き認識、ジェスチャ認識、および視覚認識のうち１つのためのものであることを特徴とする請求項１に記載のコンピュータ読取り可能媒体。

【請求項４】　文法を示す前記命令が、音声認識のための文法の参照を含むことを特徴とする請求項３に記載のコンピュータ読取り可能媒体。

【請求項５】　前記マークアップ言語が、認識された音声に関連する認識結果を受け取り、前記結果を前記クライアントデバイス上にあるデータフィールドに関連付けるための命令を含むことを特徴とする請求項１に記載のコンピュータ読取り可能媒体。

【請求項６】　認識結果を受け取るための前記命令が、前記認識結果を複数のデータフィールドに関連付けることを特徴とする請求項８に記載のコンピュータ読取り可能媒体。

【請求項７】　前記マークアップ言語が、前記音声が認識されなかったとき聞き取れるように指示するための命令を含むことを特徴とする請求項１に記載のコンピュータ読取り可能媒体。

【請求項８】　前記入力がテキスト文字列を含むことを特徴とする請求項１に記載のコンピュータ読取り可能媒体。

【請求項９】　前記マークアップ言語が、前記テキスト文字列を他の構成要素から受け取るための命令を含むことを特徴とする請求項８に記載のコンピュータ読取り可能媒体。

【請求項１０】　前記マークアップ言語が、前記テキスト文字列を処理するために前記文法を関連付けることを特徴とする請求項９に記載のコンピュータ読取り可能媒体。

【請求項１１】　実施されたときに、

入力データ用のフィールドを有するＷｅｂサーバからマークアップ言語ページを受け取るステップと、

ユーザから前記フィールドに関する入力を受け取るステップと、

前記入力を示すデータおよび認識用の文法の指示を送信するステップとを含むステップを実行することによってコンピュータに情報を処理させるコンピュータ読取り可能命令を含むことを特徴とするコンピュータ読取り可能媒体。

2

【請求項１２】　前記指示が、前記文法の場所の参照を提供することを特徴とする請求項１１に記載のコンピュータ読取り可能媒体。

【請求項１３】　前記指示が、音声認識用の言語の参照を含むことを特徴とする請求項１２に記載のコンピュータ読取り可能媒体。

【請求項１４】　クライアント／サーバネットワークでの認識のための方法であって、

クライアントデバイスとは異なるネットワーク上のアドレスにあるＷｅｂサーバから、ネットワークに接続されたクライアントデバイス上に入力データ用のフィールドを有するマークアップ言語ページを受け取るステップと、

前記クライアントデバイス上のフィールドに関するユーザから入力を受け取るステップ、および前記入力を示すデータおよび認識用の文法の指示を、前記クライアントデバイスとは異なるネットワーク上のアドレスにある認識サーバに送信するステップとを含むことを特徴とする方法。

【請求項１５】　前記データを送信する前に、前記入力を示す前記データを標準化するステップをさらに含むことを特徴とする請求項１４に記載の方法。

【請求項１６】　認識された音声に関連する認識結果を受け取り、前記結果を前記クライアントデバイス上にあるデータフィールドに関連付けるステップをさらに含むことを特徴とする請求項４１に記載の方法。

【請求項１７】　クライアント／サーバシステム中のクライアントデバイスで実行するためのマークアップ言語を有するコンピュータ読取り可能媒体であって、前記マークアップ言語が、認識用の属性または方法を有するオブジェクトモデルを示す命令を含むことを特徴とするコンピュータ読取り可能媒体。

【請求項１８】　前記要素が、音声認識、手書き認識、ジェスチャ認識、ＤＴＭＦ認識、および視覚認識のうち１つのためのものであることを特徴とする請求項１７に記載のコンピュータ読取り可能媒体。

【請求項１９】　前記要素が、認識用の文法の指示を含むことを特徴とする請求項１７に記載のコンピュータ読取り可能媒体。

【請求項２０】　前記要素が、認識などのテキスト文字列を受け取って処理することに関連することを特徴とする請求項１９に記載のコンピュータ読取り可能媒体。

【請求項２１】　前記テキスト文字列を他の構成要素から受け取る第２のオブジェクトモデルをさらに含むことを特徴とする請求項２０に記載のコンピュータ読取り可能媒体。

【請求項２２】　クライアント／サーバシステム中のクライアントデバイスで実行するためのマークアップ言語を有するコンピュータ読取り可能媒体であって、前記マークアップ言語が、聞き取れるようにプロンプティングするための属性または方法を有するオブジェクトモデルを示す命令を含むことを特徴とするコンピュータ読取り可能媒体。

3

【請求項２３】　前記マークアップ言語が、ＨＴＭＬ、ＸＨＴＭＬ、ｃＨＴＭＬ、ＸＭＬ、およびＷＭＬのうち１つを含むことを特徴とする請求項２２に記載のコンピュータ読取り可能媒体。

【請求項２４】　前記マークアップ言語が、スクリプティング言語を含むことを特徴とする請求項２２に記載のコンピュータ読取り可能媒体。

【請求項２５】　前記マークアップ言語が、合成されたマークアップ言語を含むことを特徴とする請求項２２に記載のコンピュータ読取り可能媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、インターネットなどのワイドエリアネットワークを介した情報へのアクセスに関する。より詳細には、本発明は、様々な方法を使用してクライアント側で情報および制御の入力を可能にする、Ｗｅｂで使用可能な認識に関する。

【０００２】

【従来の技術】パーソナルインフォメーションマネージャ（ＰＩＭ）などの小型コンピューティングデバイス、装置、および携帯電話は、人々の日常活動の中でますます頻繁に使用されている。これらのデバイスを作動させるマイクロプロセッサに現在使用できる処理能力が向上するにつれて、これらのデバイスの機能も向上し、場合によっては統合される。たとえば携帯電話の多くは、インターネットへのアクセスおよびブラウジングが可能であり、さらにアドレス、電話番号などの個人情報を記憶しておくのに使用することもできる。

【０００３】これらのコンピューティングデバイスが、インターネットのブラウジングに使用され、または他のサーバ／クライアントアーキテクチャで使用されることに鑑みて、情報をコンピューティングデバイスに入力する必要がある。残念ながら、これらのデバイスは、持ち運びやすいようにできる限り小型であることが望ましく、コンピューティングデバイスのハウジング上では利用可能な表面積が限られているため、すべてのアルファベット文字を独立したボタンとする従来のキーボードは、通常不可能である。

【０００４】近年では、ＶｏｉｃｅＸＭＬ（ｖｏｉｃｅ ｅｘｔｅｎｓｉｂｌｅ ｍａｒｋｕｐ ｌａｎｇｕａｇｅ）を使用するような音声ポータルが、電話のみを使用してインターネットコンテンツにアクセスできるように改良されてきた。このアーキテクチャでは、ドキュメントサーバ（たとえば、Ｗｅｂサーバ）は、ＶｏｉｃｅＸＭＬインタープリタを介してクライアントからの要求を処理する。Ｗｅｂサーバはこれに応答してＶｏｉｃｅ

4

ＸＭＬドキュメントを作成することが可能であり、これらのドキュメントは、ＶｏｉｃｅＸＭＬインタープリタによってユーザが聞き取れるように処理される。ユーザは、音声認識を介した音声コマンドを使用して、Ｗｅｂをナビゲートすることができる。

【０００５】

【発明が解決しようとする課題】ＶｏｉｃｅＸＭＬは、フロー制御タグを備えたマークアップ言語であるが、フロー制御は、イベントスクリプトおよび分離スクリプトを含むＨＴＭＬ（Ｈｙｐｅｒ Ｔｅｘｔ Ｍａｒｋｕｐ Ｌａｎｇｕａｇｅ）フロー制御モデルに従うものではない。むしろ、ＶｏｉｃｅＸＭＬは一般に、特に電話ベースの音声専用対話に好適な形態解釈アルゴリズムを含み、一般にユーザから取得された情報は、システムまたはアプリケーションの制御下に置かれる。グラフィカルユーザインターフェースも提供されているクライアントサーバ関係で使用できるアプリケーションに、ＶｏｉｃｅＸＭＬを直接組み込むには、開発者が、ＶｏｉｃｅＸＭＬ用のものとＨＴＭＬ（または同様のもの）を使用するものであり、それぞれが異なるフロー制御モデルに従っている、２つの形式のＷｅｂオーサリングを習得する必要がある。

【０００６】したがって、インターネットなどのサーバ／クライアントアーキテクチャで音声認識を提供するのに使用されるアーキテクチャおよび方法を改善することが引き続き求められている。音声認識用のオーサリングツールは、ＰＩＭや電話などの小型のコンピューティングデバイスに容易に適応できるべきである。前述の欠点のうちの１つ、いくつか、あるいはすべてに対処するＷｅｂオーサリングのアーキテクチャおよび方法が特に求められている。

【０００７】

【課題を解決するための手段】クライアント／サーバシステム中のクライアントデバイスで実行するためのマークアップ言語には、クライアントデバイスを介して入力された入力データに関連付けるための文法を示す命令が含まれる。

【０００８】この拡張部（ｅｘｔｅｎｓｉｏｎ）、および本発明の他の態様を使用できることで、クライアントデバイスは、入力データ用のフィールドを有するＷｅｂサーバからマークアップ言語ページを受け取るための命令を実行することができる。次いでクライアントデバイスは、そのフィールドに関連するユーザから入力データを受け取り、そのデータおよび認識のための文法の指示を、典型的には処理用にリモート位置に配置された認識サーバに送信する。

【０００９】認識サーバは、認識を実行するために、入力データおよび文法の指示を受け取るための命令を実行することができる。次いで、認識の結果をさらに処理するために、クライアントデバイスまたはＷｅｂサーバに

5

返信することができる。

【００１０】

【発明の実施の形態】同じ内容を実施するためのＷｅｂベースの認識およびメソッドのアーキテクチャについて説明する前に、そのアーキテクチャで機能できる一般的なコンピューティングデバイスについて説明する方が有用であろう。図１を参照すると、データ管理デバイス（ＰＩＭ、ＰＤＡなど）の形態例が３０に示されている。ただし、本発明は、以下で論じる他のコンピューティングデバイス、具体的に言えば、入力ボタン用などの表面積が限られているコンピューティングデバイスを使用しても実施可能であることが企図される。たとえば、電話および／またはデータ管理デバイスも、本発明からの恩恵を受けることになる。こうしたデバイスは、既存の携帯用個人情報管理デバイスおよび他の携帯用電子デバイスに比べて、機能強化されたユーティリティを有するものとなり、こうしたデバイスの機能およびコンパクトなサイズにより、ユーザに対して常に持ち運ぶように奨励しやすくなる。したがって、本明細書に記載されたアーキテクチャの範囲は、本明細書に示した例示的なデータ管理またはＰＩＭデバイス、電話、またはコンピュータの開示によって限定されることを意図するものではない。

【００１１】データ管理移動デバイス３０の例示的形態が、図１に示されている。移動デバイス３０はハウジング３２を含み、スタイラス３３と一緒に使われる接触感応性のディスプレイ画面を使用するディスプレイ３４を含むユーザインターフェースを有する。スタイラス３３は、フィールドを選択するために指定された座標位置でディスプレイ３４を押すかまたは接触するため、カーソルの開始位置を選択的に移動するため、あるいはその他の方法でジェスチャまたは手書きなどのコマンド情報を提供するために、使用される。別法では、または加えて、デバイス３０上にナビゲーション用の１つまたは複数のボタン３５を含めることができる。さらに、回転ホイール、ローラなどの他の入力メカニズムを提供することもできる。ただし、本発明は、これらの入力メカニズム形態によって限定されるものではないことに留意されたい。たとえば他の入力形態には、コンピュータビジョンを介するなどの視覚的な入力を含むことができる。

【００１２】次に図２を参照すると、移動デバイス３０を含む機能構成要素の構成図が示されている。中央処理ユニット（ＣＰＵ）５０は、ソフトウェア制御機能を実施する。ＣＰＵ５０がディスプレイ３４に結合されているため、制御ソフトウェアに従って生成されたテキストおよびグラフィックアイコンがディスプレイ３４に表示される。スピーカ４３は、可聴出力を提供するために、典型的にはデジタルアナログ変換器５９を使用してＣＰＵ５０に結合することができる。ユーザによってダウンロードされ、移動デバイス３０に入力されたデータは、

6

ＣＰＵ５０と双方向に結合された不揮発性読取り／書込みランダムアクセスメモリ記憶装置５４に格納される。ランダムアクセスメモリ（ＲＡＭ）５４は、ＣＰＵ５０によって実行される命令用の揮発性記憶域、およびレジスタ値などの一時データ用の記憶域を提供する。構成オプションおよび他の変数用のデフォルト値は、読取り専用メモリ（ＲＯＭ）５８に格納される。ＲＯＭ　５８は、移動体３０の基本機能および他のオペレーティングシステムのカーネル機能（たとえば、ソフトウェア構成要素のＲＡＭ５４へのローディング）を制御する、デバイス用のオペレーティングシステムソフトウェアを格納するために使用することもできる。

【００１３】ＲＡＭ５４は、アプリケーションプログラムを格納するのに使用されるＰＣ上のハードドライブの機能に類似した方法で、コード用の記憶域としての役割も果たす。コードを格納するには不揮発性メモリが使用されるが、代替方法として、コードの実行用には使用されない揮発性メモリ内に格納することもできることに留意されたい。

【００１４】無線信号は、ＣＰＵ５０に結合された無線トランシーバ５２を介して移動デバイスによって送受信することができる。コンピュータ（たとえばデスクトップコンピュータ）から、または望むなら有線ネットワークから、データを直接ダウンロードするために、オプションの通信インターフェース６０を提供することもできる。したがってインターフェース６０は、たとえば、赤外線リンク、モデム、ネットワークカードなどの様々な形態の通信デバイスを含むことができる。

【００１５】移動デバイス３０は、マイクロフォン２９およびアナログデジタル（Ａ／Ｄ）変換器３７、ならびに記憶装置５４に格納されたオプションの認識プログラム（音声、ＤＴＭＦ、手書き、ジェスチャ、またはコンピュータビジョン）を含む。例を挙げると、デバイス３０のユーザからの可聴情報、命令、またはコマンドに応答して、マイクロフォン２９が音声信号を発信し、これがＡ／Ｄ変換器３７によってデジタル化される。音声認識プログラムは、音声認識中間結果を取得するために、デジタル化された音声信号に対して、正規化および／または特徴抽出機能を実行することができる。音声データは、無線トランシーバ５２または通信インターフェース６０を使用して、以下で考察し図５のアーキテクチャに示された、リモート認識サーバ２０４に伝送される。その後認識結果は、提示し（たとえば視覚的および／または聴覚的）、かつ結果としてＷｅｂサーバ２０２（図５）に伝送するために移動デバイス３０に戻され、Ｗｅｂサーバ２０２および移動デバイス３０はクライアント／サーバの関係で動作する。同様の処理が、他の入力形態用に使用される。たとえば手書き入力は、デバイス３０上での事前処理によって、または事前処理なしで、デジタル化することができる。音声データと同様に、この

7

形態の入力を認識のために認識サーバ２０４に伝送し、認識結果をデバイス３０および／またはＷｅｂサーバ２０２のうち少なくとも１つに戻すことができる。同様に、ＤＴＭＦデータ、ジェスチャデータ、および視覚データを同じように処理することができる。入力形態に応じて、デバイス３０（および以下で論じる他の形態のクライアント）には、視覚入力用のカメラなどの必要なハードウェアが含まれることになる。

【００１６】図３は、携帯電話８０の例示的実施形態を示す平面図である。電話８０にはディスプレイ８２およびキーパッド８４が含まれる。一般に図２の構成図は図３の電話に適用されるが、他の機能を実行するのに必要な追加の回路を要求することができる。たとえば、図２の実施形態では、電話として動作することが必要なトランシーバが要求されるが、こうした回路は本発明には関係していない。

【００１７】前述の携帯または移動コンピューティングデバイスに加えて、本発明が、汎用デスクトップコンピュータなどの他の多数のコンピューティングデバイスで使用可能であることも理解されたい。たとえば本発明は、身体的能力の制限されたユーザが、フル英数字キーボードなどの他の従来型入力デバイスを操作するのが困難である場合に、テキストをコンピュータまたは他のコンピューティングデバイスに入力できるようにするものである。

【００１８】本発明は、他の多数の汎用または特定用途向けコンピューティングシステム、環境、または構成でも動作可能である。本発明で使用するのに好適な、よく知られたコンピューティングシステム、環境、および／または構成の例には、通常の電話（画面なし）、パーソナルコンピュータ、サーバコンピュータ、ハンドヘルドまたはラップトップデバイス、マルチプロセッサシステム、マイクロプロセッサベースシステム、セットトップボックス、プログラム可能大衆消費電子製品、ネットワークＰＣ、ミニコンピュータ、メインフレームコンピュータ、上記システムまたはデバイスのいずれかを含む分散コンピューティング環境などが含まれるが、これらに限定されるものではない。

【００１９】次に、図４に示した汎用コンピュータ１２０について、簡単に説明する。ただし、コンピュータ１２０も好適なコンピューティング環境の一例に過ぎず、本発明の使用法または機能の範囲に関して何らかの制限を示唆するものではない。さらにコンピュータ１２０は、本明細書に例示された構成要素のいずれか１つまたはいずれかの組合せに関して、なんらかの依存性または要求条件を有するものとして解釈されるものでもない。

【００２０】本発明は、コンピュータによって実行されるプログラムモジュールなどの、コンピュータ実行可能命令の一般的な文脈で説明することができる。一般に、プログラムモジュールには、特定のタスクを実行するか

8

または特定の抽象データ型を実施する、ルーチン、プログラム、オブジェクト、構成要素、データ構造などが含まれる。本発明は、通信ネットワークを介してリンクされたリモート処理デバイスによってタスクが実施される、分散コンピューティング環境で実施することも可能である。分散コンピューティング環境では、プログラムモジュールを、メモリ記憶デバイスを含む、ローカルおよびリモートの両方のコンピュータ記憶媒体内に配置することが可能である。プログラムおよびモジュールによって実行されるタスクについては、以下で図を用いて説明する。当分野の技術者であれば、記述および図面を、任意の形態のコンピュータ読取り可能媒体上に書き込み可能な処理装置実行可能命令として実施することができる。

【００２１】図４を参照すると、コンピュータ１２０の構成要素には、処理ユニット１４０、システムメモリ１５０、およびシステムメモリを含む様々なシステム構成要素を処理ユニット１４０に結合するシステムバス１４１が含まれるが、これらに限定されるものではない。システムバス１４１は、メモリバスまたはメモリ制御装置、周辺バス、および様々なバスアーキテクチャのいずれかを使用するローカルバスを含む、いくつかのタイプのバスストラクチャのうちのいずれかであってよい。例を挙げると、こうしたアーキテクチャには、Ｉｎｄｕｓｔｒｙ　Ｓｔａｎｄａｒｄ　Ａｒｃｈｉｔｅｃｔｕｒｅ（ＩＳＡ）バス、Ｕｎｉｖｅｒｓａｌ　Ｓｅｒｉａｌ　Ｂｕｓ（ＵＳＢ）、Ｍｉｃｒｏ　Ｃｈａｎｎｅｌ　Ａｒｃｈｉｔｅｃｔｕｒｅ（ＭＣＡ）バス、拡張ＩＳＡ（ＥＩＳＡ）バス、Ｖｉｄｅｏ　Ｅｌｅｃｔｒｏｎｉｃｓ　Ｓｔａｎｄａｒｄｓ　Ａｓｓｏｃｉａｔｉｏｎ（ＶＥＳＡ）ローカルバス、およびメザニンバスとも呼ばれるＰｅｒｉｐｈｅｒａｌ　Ｃｏｍｐｏｎｅｎｔ　Ｉｎｔｅｒｃｏｎｎｅｃｔ（ＰＣＩ）バスが含まれるが、これらに限定されるものではない。コンピュータ１２０は、典型的には様々なコンピュータ読取り可能媒体を含む。コンピュータ読取り可能媒体は、コンピュータ１２０がアクセスすることのできる任意の使用可能媒体であってよく、揮発性および不揮発性媒体、取外し可能および取外し不能媒体の両方を含む。例を挙げると、コンピュータ読取り可能媒体は、コンピュータ記憶媒体および通信媒体を含むことができるが、これらに限定されるものではない。コンピュータ記憶媒体には、コンピュータ読取り可能命令、データ構造、プログラムモジュール、または他のデータなどの情報を記憶するための任意の方法または技術で実施される、揮発性および不揮発性、取外し可能および取外し不能の両方の媒体が含まれる。コンピュータ記憶媒体には、ＲＡＭ、ＲＯＭ、ＥＥＰＲＯＭ、フラッシュメモリまたは他のメモリ技術、ＣＤ－ＲＯＭ、デジタル汎用ディスク（ＤＶＤ）または他の光ディスク記憶装置、磁気カセット、磁気テープ、磁気ディスク記

9

憶装置または他の磁気記憶デバイス、あるいは、所望の情報を格納するのに使用可能でありコンピュータ１２０がアクセス可能な任意の他の媒体が含まれるが、これらに限定されるものではない。

【００２２】通信媒体は、典型的には、搬送波または他の移送メカニズムなどの被変調データ信号中で、コンピュータ読取り可能命令、データ構造、プログラムモジュール、または他のデータを具体化するものであり、任意の情報送達媒体を含む。「被変調データ信号」という用語は、１つまたは複数の特徴セットを有するか、または信号内での情報の符号化などの様式が変更された信号を意味するものである。例を挙げると、通信媒体には、有線ネットワークまたは直接配線接続などの有線媒体、ならびに音波、ＦＲ、赤外線、および他の無線媒体などの無線媒体が含まれるが、これらに限定されるものではない。上記の任意の組合せも、コンピュータ読取り可能媒体の範囲に含まれるものである。

【００２３】システムメモリ１５０には、読取り専用メモリ（ＲＯＭ）１５１およびランダムアクセスメモリ（ＲＡＭ）１５２などの、揮発性および／または不揮発性メモリの形態のコンピュータ記憶媒体が含まれる。起動時などのコンピュータ１２０内の要素間での情報転送に役立つ基本ルーチンを含む、基本入出力システム１５３（ＢＩＯＳ）は、典型的にはＲＯＭ　１５１に格納される。ＲＡＭ　１５２は典型的には、即時アクセス可能であるか、あるいは／ならびに現在処理ユニット１４０によって動作中である、データおよび／またはプログラムモジュールを含む。例を挙げると、図４ではオペレーティングシステム５４、アプリケーションプログラム１５５、他のプログラムモジュール１５６、およびプログラムデータ１５７が示されているが、これらに限定されるものではない。

【００２４】コンピュータ１２０は、他の取外し可能／取外し不能、揮発性／不揮発性のコンピュータ記憶媒体を含むこともできる。図４では、取外し不能の不揮発性磁気媒体との間で読取りまたは書込みを行うハードディスクドライブ１６１、取外し可能の不揮発性磁気ディスク１７２との間で読取りまたは書込みを行う磁気ディスクドライブ１７１、ならびにＣＤ　ＲＯＭまたは他の光学式媒体などの取外し可能の不揮発性光ディスク１７６との間で読取りまたは書込みを行う光ディスクドライブ１７５が示されているが、これらは例として示したものにすぎない。例示的な動作環境で使用することのできる他の取外し可能／取外し不能、揮発性／不揮発性コンピュータ記憶媒体には、磁気テープカセット、フラッシュメモリカード、デジタル汎用ディスク、デジタルビデオテープ、ソリッドステートＲＡＭ、ソリッドステートＲＯＭ、などが含まれるが、これらに限定されるものではない。ハードディスクドライブ１６１は、典型的には、インターフェース１６０などの取外し不能メモリインターフェースを介してシステムバス１４１に接続され、磁気ディスクドライブ１７１および光ディスクドライブ１７５は、典型的には、インターフェース１７０などの取外し可能メモリインターフェースによってシステムバス１４１に接続される。

10

【００２５】上記で考察し図４に示したドライブおよびその関連付けられたコンピュータ記憶媒体は、コンピュータ１２０に、コンピュータ読取り可能命令、データ構造、プログラムモジュール、および他のデータの記憶域を提供する。たとえば図４では、ハードディスクドライブ１６１がオペレーティングシステム１６４、アプリケーションプログラム１６５、他のプログラムモジュール１６６、およびプログラムデータ１６７を格納しているように図示されている。これらの構成要素は、オペレーティングシステム１５４、アプリケーションプログラム１５５、他のプログラムモジュール１５６、およびプログラムデータ１５７と同じである場合も異なる場合も、いずれも可能であることに留意されたい。本明細書では、オペレーティングシステム１６４、アプリケーションプログラム１６５、他のプログラムモジュール１６６、およびプログラムデータ１６７には、少なくとも異なるコピーであることを示すために異なる番号が与えられている。

【００２６】ユーザは、キーボード１８２、マイクロフォン１８３、およびマウス、トラックボール、またはタッチパッドなどのポインティングデバイス１８１などの入力デバイスを介して、コマンドおよび情報をコンピュータ１２０に入力することができる。他の入力デバイス（図示せず）には、ジョイスティック、ゲームパッド、衛星放送用アンテナ、スキャナ、などが含まれる。これらおよび他の入力デバイスは、システムバスに結合されたユーザ入力インターフェース１８０を介して処理ユニット１４０に接続される場合が多いが、パラレルポート、ゲームポート、またはＵｎｉｖｅｒｓａｌ　Ｓｅｒｉａｌ　Ｂｕｓ（ＵＳＢ）などの他のインターフェースおよびバス構造によって接続することもできる。モニタ１８４または他のタイプのディスプレイデバイスも、ビデオインターフェース１８５などのインターフェースを介してシステムバス１４１に接続される。コンピュータは、モニタに加えて、出力周辺インターフェース１８８を介して接続することのできるスピーカ１８７およびプリンタ１８６などの他の周辺出力デバイスも含むことができる。

【００２７】コンピュータ１２０は、リモートコンピュータ１９４などの１つまたは複数のリモートコンピュータへの論理接続を使用するネットワーク環境で動作可能である。リモートコンピュータ１９４は、パーソナルコンピュータ、ハンドヘルドデバイス、サーバ、ルータ、ネットワークＰＣ、ピアデバイス、または他の共通ネットワークノードであってよく、典型的にはコンピュータ

11

１２０に関連して上記で述べた要素の多くまたはすべてを含む。図４に示された論理接続には、ローカルエリアネットワーク（ＬＡＮ）１９１およびワイドエリアネットワーク（ＷＡＮ）１９３が含まれるが、他のネットワークを含むこともできる。こうしたネットワーキング環境は、オフィス、企業全体のコンピュータネットワーク、イントラネット、およびインターネットなどでよく見られるものである。

【００２８】ＬＡＮネットワーキング環境で使用される場合、コンピュータ１２０はネットワークインターフェースまたはアダプタ１９０を介してＬＡＮ　１９１に接続される。ＷＡＮネットワーキング環境で使用される場合、コンピュータ１２０は、典型的にはモデム１９２、またはインターネットなどのＷＡＮ　１９３を介して通信を確立するための他の手段を含む。モデム１９２は内蔵型または外付けが可能であり、ユーザ入力インターフェース１８０または他の適切な機構を介してシステムバス１４１に接続することができる。ネットワーク環境では、コンピュータ１２０に関連して示されたプログラムモジュール、またはその一部を、リモートのメモリ記憶デバイスに格納することができる。例を挙げると、図４ではリモートアプリケーションプログラム１９５がリモートコンピュータ１９４上に常駐しているように示されているが、これに限定されるものではない。図示されたネットワーク接続は例示的なものであり、コンピュータ間で通信リンクを確立するための他の手段も使用可能であることを理解されよう。

【００２９】図５は、本発明で実施可能な、Ｗｅｂベース認識用のアーキテクチャ２００を示す図である。一般に、Ｗｅｂサーバ２０２に格納された情報には、移動デバイス３０（本明細書では、入力の形態に基づき、必要に応じて、ディスプレイ画面、マイクロフォン、カメラ、タッチセンシティブパネルなどを有するコンピューティングデバイスの他の形態も表している）、または電話８０を介してアクセスすることが可能であり、情報は、聞き取れるように要求されるか、またはキーが押されるのに応答して電話８０が生成するトーンによって要求され、Ｗｅｂサーバ２０２からの情報は、必ずユーザに聞き取れるように返信される。

【００３０】さらに重要なことに、アーキテクチャ２００は、情報がデバイス３０と音声認識を使用する電話８０のどちらかを介して得られるように統合されているが、単一の認識サーバ２０４はどちらのモードの動作もサポートすることができる。さらに、アーキテクチャ２００はよく知られたマークアップ言語（たとえばＨＴＭＬ、ＸＨＴＭＬ、ｃＨＴＭＬ、ＸＭＬ、ＷＭＬなど）の拡張部を使用して動作する。したがって、Ｗｅｂサーバ２０２に格納された情報には、これらのマークアップ言語に見られるよく知られたＧＵＩ方式を使用してアクセスすることもできる。よく知られたマークアップ言語の

12

拡張部を使用することで、Ｗｅｂサーバ２０２でのオーサリングが容易になり、現在存在しているレガシーアプリケーションを音声認識を含むように修正することも容易となる。

【００３１】一般に、デバイス３０は、Ｗｅｂサーバ２０２が提供するＨＴＭＬページ、スクリプトなどを実行する。例を挙げると、音声認識が要求される場合、可聴信号にデジタル化することができる音声データ、または前述のようにデバイス３０によって可聴信号が事前処理されている音声特徴が、音声認識時に使用するための文法または言語モデルの指示と共に、認識サーバ２０４に提供される。認識サーバ２０４の実施は多くの形態を取ることが可能であり、そのうちの１つが図示されているが、一般には認識装置２１１が含まれる。認識の結果は、所望されるかまたは適切であれば、ローカルで提示するためにデバイス３０に戻される。認識および供されている場合は任意のグラフィカルユーザインターフェースを介して情報をコンパイルすると、デバイス３０は、その情報をＷｅｂサーバ２０２に送信してさらに処理し、必要であればさらにＨＴＭＬページ／スクリプトを受け取る。

【００３２】図５に示されるように、デバイス３０、Ｗｅｂサーバ２０２、および認識サーバ２０４は、一般にネットワーク２０５を介して接続されるが、本明細書ではインターネットなどのワイドエリアネットワークを介して接続されており、別々にアドレス指定可能である。したがって、これらのデバイスはいずれも物理的に相互に隣接させる必要はない。具体的に言えば、Ｗｅｂサーバ２０２が認識サーバ２０４を含む必要はない。この方式では、Ｗｅｂサーバ２０２でのオーサリングは、作者が認識サーバ２０４の複雑な内容を知る必要なく、意図されているアプリケーションに焦点をあてることが可能である。むしろ認識サーバ２０４は、単独で設計してネットワーク２０５に接続することが可能であり、それによって、Ｗｅｂサーバ２０２で要求される変更なしに、更新および改良することが可能である。以下で論じるように、Ｗｅｂサーバ２０２は、動的にクライアント側のマークアップおよびスクリプトを生成することができるオーサリング機構を含むこともできる。他の実施形態では、Ｗｅｂサーバ２０２、認識サーバ２０４、およびクライアント３０を、実施マシンの機能に応じて組み合わせることができる。たとえば、クライアントが汎用コンピュータ、たとえばパーソナルコンピュータを含む場合、クライアントは認識サーバ２０４を含むことができる。同様に、所望であれば、Ｗｅｂサーバ２０２および認識サーバ２０４を単一のマシンに組み込むことができる。

【００３３】本発明の一態様は、クライアントデバイスのユーザから入力データを取得するように構成された拡張部を有するマークアップ言語ページをサーバから受け

取り、クライアントデバイス上でマークアップ言語ページを実行し、入力データ（ユーザから取得した音声、ＤＴＭＦ、手書き、ジェスチャ、またはイメージを示す）および関連付けられた文法をクライアントから遠隔にある認識サーバに伝送し、ならびに、クライアント側の認識サーバから認識結果を受け取るステップを含む、クライアント／サーバシステムで入力データを処理するための方法である。他の態様は、クライアント／サーバシステムのクライアントデバイスで実行するためのマークアップ言語を有するコンピュータ読取り可能媒体であって、このマークアップ言語は、クライアントデバイスを介して入力された入力データに関連付けるための文法を示す命令を有する。

【００３４】電話８０を介したＷｅｂサーバ２０２へのアクセスには、電話８０から有線または無線電話ネットワーク２０８への接続が含まれ、これが電話８０を第三者ゲートウェイ２１０に接続する。ゲートウェイ２１０は、電話８０を電話音声ブラウザ２１２に接続する。電話音声ブラウザ２１２には、電話インターフェースおよび音声ブラウザ２１６を提供するメディアサーバ２１４が含まれる。デバイス３０と同様に、電話音声ブラウザ２１２はＨＴＭＬページ／スクリプトなどをＷｅｂサーバ２０２から受け取る。さらに重要なことに、ＨＴＭＬページ／スクリプトは、デバイス３０に提供されるＨＴＭＬページ／スクリプトと同様の形式である。この方式では、Ｗｅｂサーバ２０２がデバイス３０および電話８０を別々にサポートする必要がないか、または標準のＧＵＩクライアントを別々にサポートする必要もない。むしろ、通常のマークアップ言語を使用することができる。さらに、デバイス３０と同様に、電話８０によって伝送される可聴信号からの音声認識が、ネットワーク２０５または専用回線２０７のいずれかを介して、たとえばＴＣＰ／ＩＰを使用して、音声ブラウザ２１６から認識サーバ２０４に提供される。Ｗｅｂサーバ２０２、認識サーバ２０４、および電話音声ブラウザ２１２は、図４に示された汎用デスクトップコンピュータなどの任意の好適なコンピューティング環境で実施することができる。

【００３５】ただし、ＤＴＭＦ認識が使用される場合、この形態の認識は一般に、認識サーバ２０４ではなくメディアサーバ２１４で実行されることに留意されたい。言い換えれば、ＤＴＭＦ文法はメディアサーバによって使用される。

【００３６】前述のように、本発明の一態様は、クライアント／サーバアーキテクチャに認識を実現する制御および／またはオブジェクトを含めるための、ＨＴＭＬ、ＸＨＴＭＬ、ｃＨＴＭＬ、ＸＭＬ、ＷＭＬなどのマークアップ言語の拡張部を含むか、または任意の他のＳＧＭＬ導出型マークアップを備えている。この方式では、作者が、こうしたアーキテクチャで使用される有力なＷｅｂ開発プラットフォームであるこれらのマークアップ言語で、すべてのツールおよび専門技術を活用することができる。

【００３７】一般に、制御および／またはオブジェクトは、認識装置の構成、認識装置の実行、および／または後処理のための、認識装置の制御および／またはオブジェクトと、合成装置の構成およびプロンプトの再生のための合成装置の制御および／またはオブジェクトと、入力文法リソースを指定するための文法の制御および／またはオブジェクトと、ならびに／あるいは、認識結果を処理するための結合の制御および／またはオブジェクトという機能のうち、１つまたは複数を含むことができる。拡張部は、可聴、視覚、手書きなどのインターフェースの能力を既存のマークアップ言語に追加する、軽量マークアップ層となるように設計される。したがって拡張部を、それらが含まれるたとえばＨＴＭＬなどの上位ページ、拡張部が言語リソースを参照するのに使用した、テキストから音声への変換フォーマットおよび文法フォーマットなどの下位フォーマット、ならびに認識サーバ２０４で使用される認識および音声合成プラットフォームの個々のプロパティから、独立させておくことができる。

【００３８】認識に好適な制御および／またはオブジェクトを有するマークアップ言語について述べる前に、本明細書でＨＴＭＬマークアップ言語を使用して実施される、簡単なＧＵＩの例を検討することが有用であろう。図６を参照すると、オンライン販売を完了させるためにクレジットカード情報をＷｅｂサーバに提供することを含む、簡単なＧＵＩインターフェースが示されている。この例では、クレジットカード情報に、たとえばＶｉｓａ、ＭａｓｔｅｒＣａｒｄ、またはＡｍｅｒｉｃａｎ Ｅｘｐｒｅｓｓなど、使用されるクレジットカードのタイプを入力するためのフィールド２５０が含まれる。第２のフィールド２５２はクレジットカード番号を入力することが可能であり、第３のフィールド２５４は有効期限を入力することが可能である。フィールド２５０、２５２、および２５４に入力された情報を伝送するために、サブミット（ｓｕｂｍｉｔ）ボタン２６４が提供されている。

【００３９】図７は、クライアントから前述のクレジットカード情報を取得するための、ＨＴＭＬコードを示す図である。一般に、これらのマークアップ言語の形態に共通するように、コードには本文部分２６０およびスクリプト部分２６２が含まれる。本文部分２６０には、実行される動作のタイプ、使用する形態、情報の様々なフィールド２５０、２５２、および２５４を示す複数行のコード、ならびにサブミットボタン２６４のコードが含まれる（図６）。この例では、イベンティング（ｅｖｅｎｔｉｎｇ）サポートおよび埋め込まれたスクリプトホスティングも示されており、サブミットボタン２６４を

起動すると、関数「ｖｅｒｉｆｙ」が呼び出される、またはスクリプト部分２６２で実行される。関数「ｖｅｒｉｆｙ」は、各クレジットカード（Ｖｉｓａ、ＭａｓｔｅｒＣａｒｄ、およびＡｍｅｒｉｃａｎ Ｅｘｐｒｅｓｓ）のカード番号が適切な長さであるかどうかを確認する。

【００４０】図８は、音声認識を使用してＷｅｂサーバ２０４に提供されるクレジットカード情報を取得するために、図６のＧＵＩと同じものを生成するクライアントマークアップを示す図である。音声認識については、図８～図１６に関して以下で論じるが、説明する技術は、手書き認識、ジェスチャ認識、およびイメージ認識で同様に適用可能であることを理解されたい。

【００４１】一般に、拡張部（一般に「タグ」とも呼ばれる）は、関連付けられた属性およびＤＯＭオブジェクトプロパティを備えたＸＭＬ要素と、認識インターフェースを適用するためのソースマークアップドキュメントに関して使用することができるイベントおよびメソッドと、ソースページへのＤＴＭＦまたは呼制御とのスモールセットである。拡張部の形式および意味は、ソースドキュメントの性質とは無関係であるため、拡張部は、ＨＴＭＬ、ＸＨＴＭＬ、ｃＨＴＭＬ、ＸＭＬ、ＷＭＬ内で、または任意の他のＳＧＭＬ導出型マークアップを使用して、等しく有効に使用することができる。拡張部は、階層状であってよい新しい機能オブジェクトまたは要素が提供される、ドキュメントオブジェクトモデルの後に付けられる。各要素については付録で詳細に論じるが、一般に要素は、属性、プロパティ、メソッド、イベント、および／または他の「子」要素を含むことができる。

【００４２】ここで、拡張部は、ブラウザが実行されるデバイスの機能に従って、２つの異なる「ｍｏｄｅ」で解釈できることにも留意されたい。第１のモード「ｏｂｊｅｃｔ ｍｏｄｅ」では、全機能が使用可能である。アプリケーションによる拡張部のプログラム操作には、たとえば、ＸＨＴＭＬブラウザのＪＳｃｒｉｐｔインタープリタ、またはＷＭＬブラウザのＷＭＬＳｃｒｉｐｔインタープリタなど、デバイス上のブラウザによって使用可能にされるどのような機構によっても実行される。そのため、拡張部のコアプロパティおよびメソッドのスモールセットだけを定義すればよく、これらは、デバイスまたはクライアント側に存在するどのようなプログラム機構によっても操作される。ｏｂｊｅｃｔ ｍｏｄｅは、イベンティングおよびスクリプティングを提供しており、音声対話を介して、より細かなクライアント側制御をダイアログ作者に与えるためのより優れた機能を提供することができる。本明細書で使用される場合、全イベントおよびスクリプティングをサポートしているブラウザは「アップレベルブラウザ（ｕｐｌｅｖｅｌ ｂｒｏｗｓｅｒ）」と呼ばれる。この形態のブラウザは、拡

張部のすべての属性、プロパティ、メソッド、およびイベントをサポートする。アップレベルブラウザは、一般に、より優れた処理機能を備えたデバイスに見られる。

【００４３】拡張部は、「ｄｅｃｌａｒａｔｉｖｅ ｍｏｄｅ」でもサポートすることができる。本明細書で使用される場合、ｄｅｃｌａｒａｔｉｖｅ ｍｏｄｅで動作するブラウザは「ダウンレベルブラウザ（ｄｏｗｎｌｅｖｅｌ ｂｒｏｗｓｅｒ）」と呼ばれ、イベンティングおよびスクリプティング機能をすべてサポートしているわけではない。むしろこの形態のブラウザは、所与の拡張部の宣言的な面（すなわちコア要素および属性）をサポートするが、すべてのＤＯＭ（ドキュメントオブジェクトモデル）オブジェクトのプロパティ、メソッド、およびイベントではない可能性がある。このモードは宣言型構文のみを使用し、さらに、ＳＭＩＬ（Ｓｙｎｃｈｒｏｎｉｚｅｄ Ｍｕｌｔｉｍｅｄｉａ Ｉｎｔｅｇｒａｔｉｏｎ Ｌａｎｇｕａｇｅ）２．０などの、宣言型マルチメディア同期化および調整機構（同期化されたマークアップ言語）と共に使用される場合がある。ダウンレベルブラウザは、典型的には処理機能が制限されたデバイスに見られる。

【００４４】ここで、特定モードの入力について論じなければならない。具体的に言えば、音声認識は、データ入力用のフィールドを示すために、少なくともディスプレイと共に、他の実施形態ではさらにポインティングデバイスと共に使用することが特に有用である。具体的に言えば、このモードのデータ入力では、ユーザは一般に、フィールドを選択する時期および対応する情報を提供する時期が制御される。たとえば、図６の例では、ユーザは第１にフィールド２５２にクレジットカード番号を入力し、次にフィールド２５０にクレジットカードのタイプを入力した後、フィールド２５４に有効期限を入力すると決めることができる。同様に、ユーザは、所望であればフィールド２５２に戻って、誤った入力を修正することができる。以下で論じるように、音声認識と組み合わせると、簡単で自然な形態のナビゲーションが提供される。本明細書で使用される場合、自由なフィールド選択形態を可能にするスクリーンディスプレイと音声認識の両方を使用するこの入力形態は、「ｍｕｌｔｉ－ｍｏｄａｌ」と呼ばれる。

【００４５】再度図８を参照すると、ＨＴＭＬマークアップ言語コードの例が示されている。図７に示されたＨＴＭＬコードと同様に、このコードにも本文部分２７０およびスクリプト部分２７２が含まれる。さらに図７に示されたコードと同様に、図８に示されたコードにも、実行する動作のタイプならびに形態の場所に関する指示が含まれる。各フィールド２５０、２５２、および２５４への情報の入力は、それぞれコード部分２８０、２８２、および２８４によって制御または実行される。最初にコード部分２８０を参照し、たとえばデバイス３０の

17

スタイラス３３を使用してフィールド２５０を選択すると、イベント「ｏｎＣｌｉｃｋ」が起動され、これがスクリプト部分２７２で関数「ｔａｌｋ」を呼び出すかまたは実行する。この動作により、通常はフィールド２５０で予想されるデータのタイプに関連付けられた音声認識に使用される文法が起動する。このタイプの対話が「ｍｕｌｔｉｍｏｄａｌ」と呼ばれ、複数の入力技法（たとえば音声およびペンクリック／ローラ）が含まれる。

【００４６】多くのアプリケーションで、ソースページでアプリケーション指定のグラフィック機構を使用することによって、作者がページの様々な構成要素を音声化できるようにする信号を発すると想定されるため、図８に例示された音声認識拡張部は、クライアントのブラウザ上でデフォルト時に視覚表現することを意図したものではないことに留意されたい。それにもかかわらず、視覚表現が望まれる場合は、拡張部をそのように修正することができる。

【００４７】再度文法に戻ると、文法は、文脈自由文法、Ｎ文法、またはハイブリッド文法などであるがこれらに限定されない、構文文法である。（もちろん、対応する認識形態が使用されるときには、ＤＴＭＦ文法、手書き文法、ジェスチャ文法、およびイメージ文法が使用される。）本明細書では、「文法」には、認識を実行するための情報が含まれ、他の実施形態では、たとえば特定のフィールドに入力されると予想される入力に対応する情報が含まれる。マークアップ言語の第１の拡張部を含む新しい制御２９０（本明細書では「ｒｅｃｏ」として識別）には様々な要素が含まれるが、そのうちの２つ、すなわち文法要素「ｇｒａｍｍａｒ」および「ｂｉｎｄ」要素が図示されている。一般に、Ｗｅｂサーバ２０２からクライアントにダウンロードされるコードと同様に、文法はＷｅｂサーバ２０２側から発信され、クライアントにダウンロードされるか、あるいは／ならびに音声処理のためにリモートサーバに転送されることが可能である。その後文法は、そこでキャッシュにローカルに格納することができる。最終的に文法は、認識時に使用するために認識サーバ２０４に送られる。ｇｒａｍｍａｒ要素は、属性を使用するインラインまたは参照済みのいずれかの文法を指定するのに使用される。

【００４８】認識された音声、手書き、ジェスチャ、イメージなどに対応する認識サーバ２０４からの認識結果を受け取ると直ちに、ｒｅｃｏ制御２９０の構文が、対応する結果を受け取り、対応するフィールドに関連付けるように提供され、フィールドは、ディスプレイ３４上でその中のテキストを提示することを含むことができる。例示された実施形態では、結果をクライアントに送信して音声認識が完了すると、ｒｅｃｏオブジェクトを非活動化し、認識されたテキストと対応するフィールドを関連付ける。部分２８２および２８４が同様に動作

18

し、各フィールド２５２および２５４に一意のｒｅｃｏオブジェクトおよび文法が要求され、認識されたテキストを受け取ると、フィールド２５２および２５４それぞれに関連付けられる。カード番号フィールド２５２の受取りに関しては、図７に関して上記で述べたものと同様の方法で、関数「ｈａｎｄｌｅ」がカードタイプを基準にカード番号の長さをチェックする。

【００４９】一般に、アーキテクチャ２００およびクライアント側マークアップ言語と共に音声認識を使用する場合、第１に、与えられる音声に関連付けられたフィールドが示される。例示された実施形態では、スタイラス３３が使用されるが、本発明は、スタイラス３３の使用に限定されるものではなく、ボタン、マウスポインタ、回転ホイールなどの任意の形態の指示が使用できることを理解されたい。「ｏｎＣｌｉｃｋ」などの対応するイベントを、視覚マークアップ言語の使用でよく知られるように提供することができる。本発明は、音声、手書き、ジェスチャなどのコマンドの開始を示すための「ｏｎＣｌｉｃｋ」イベントの使用に限定されるものではないことを理解されたい。「ｏｎＳｅｌｅｃｔ」など、同じ目的には、任意の使用可能なＧＵＩイベントが使用できる。一実施形態では、こうしたイベンティングが、対応する音声の開始および／または終了の両方を示す働きをするのに特に有用である。ユーザならびにユーザの対話を追跡するブラウザ上で実行中のプログラムによって、音声の宛先となるフィールドを示すことができることにも留意されたい。

【００５０】ここで、様々な音声認識シナリオが、認識サーバ２０４からの様々な動作および／または出力を必要とすることについて説明する。認識プロセスの開始はすべての場合、たとえばアップレベルブラウザからの明示的なｓｔａｒｔ（）コール、またはダウンレベルブラウザでの宣言型＜ｒｅｃｏ＞要素の場合に、基準であるが、音声認識を停止する手段は異なる場合がある。

【００５１】前述の例では、マルチモーダルアプリケーションのユーザは、たとえば、圧力感知ディスプレイ上を軽く叩き保持することによって、デバイスへの入力が制御される。次いでブラウザは、ＧＵＩイベント、たとえば「ｐｅｎ－ｕｐ」を使用して、認識を停止しなければならない時期を制御し、対応する結果を戻す。ただし、電話アプリケーション（以下で考察）またはハンズフリーアプリケーションなどの音声専用シナリオでは、ユーザがブラウザを介して直接制御することがないため、認識サーバ２０４またはクライアント３０は、認識を停止して結果を戻す時期（典型的には文法を介したパスが認識されたとき）を決定する役目を果たさなければならない。さらに、認識が停止される前に中間結果を戻す必要のある口述（ｄｉｃｔａｔｉｏｎ）および他のシナリオは（「オープンマイクロフォン」としても知られる）、明示的な停止機能を必要とするだけでなく、認識

19

プロセスが停止される前に、複数の認識結果をクライアント３０および／またはＷｅｂサーバ２０２に戻す必要もある。

【００５２】一実施形態では、Ｒｅｃｏ要素は、どのようにいつ結果を戻すかを認識サーバ２０４に指示する、以下の３つの認識モードを区別するための「ｍｏｄｅ」属性を含むことができる。結果を戻すことは、適切な「ｏｎＲｅｃｏ」イベントの提供または「ｂｉｎｄ」要素の起動を暗に示すものである。一実施形態では、モードが指定されていない場合のデフォルトの認識モードは「ａｕｔｏｍａｔｉｃ」となる。

【００５３】図１４は、音声認識のための「ａｕｔｏｍａｔｉｃ」モードの動作を示す絵画図である（同様のモード、イベントなどを他の形態の認識に提供することができる）。時間線２８１は、認識サーバ２０４が２８３で認識を開始するように指示された時点、および認識サーバ２０４が２８５で音声を検出し、２８７で音声が終了したことを特定した場所を示すものである。

【００５４】Ｒｅｃｏ要素の様々な属性は、認識サーバ２０４の動作を制御する。属性「ｉｎｉｔｉａｌＴｉｍｅｏｕｔ」２８９は、認識の開始２８３から音声の検出２８５までの時間である。この時間枠を超えると、認識が停止した旨を伝える「ｏｎＳｉｌｅｎｃｅ」イベント２９１が認識サーバ２０４から提供される。認識サーバ２０４が認識できない発話を見つけると、「ｏｎＮｏＲｅｃｏ」イベント２９３が発行されるが、これは認識が停止したことも示す。

【００５５】認識を停止または取り消すことができる他の属性に「ｂａｂｂｌｅＴｉｍｅｏｕｔ」属性２９５が含まれるが、これは認識サーバ２０４が２８５で音声を検出した後に結果を戻さなければならない時間枠である。この時間枠を超えると、エラーが生じたかどうかに応じて、異なるイベントが発行される。認識サーバ２０４が依然として音声を処理している場合、たとえば非常に長い発話の場合、「ｏｎＮｏＲｅｃｏ」属性２９３が発行される。ただし、任意の他の理由で「ｂａｂｂｌｅＴｉｍｅｏｕｔ」属性２９５を超えた場合、認識装置のエラーである可能性が高く、「ｏｎＴｉｍｅｏｕｔ」イベント２９７が発行される。同様に、「ｍａｘＴｉｍｅｏｕｔ」属性２９９も提供することが可能であり、これは認識の開始２８３から結果がクライアント３０に戻されるまでの時間枠に関するものである。この時間枠を超えると、「ｏｎＴｉｍｅｏｕｔ」イベント２９７が発行される。

【００５６】ただし、「ｅｎｄＳｉｌｅｎｃｅ」属性３０１よりも長い時間枠を超えた場合は、認識が完了したことを暗に示すものであり、認識サーバ２０４は認識を自動的に停止して、その結果を戻す。認識サーバ２０４は、認識結果を戻すべきであるかどうかを判定するための確信測定（ｃｏｎｆｉｄｅｎｃｅ　ｍｅａｓｕｒｅ）

20

を実施できることに留意されたい。確信測定がしきい値よりも低い場合は「ｏｎＮｏＲｅｃｏ」属性２９３が発行され、確信測定がしきい値よりも高い場合は「ｏｎＲｅｃｏ」属性３０３および認識結果が発行される。図１４は、「ａｕｔｏｍａｔｉｃ　ｍｏｄｅ」で、明示的なｓｔｏｐ（）コールが実行されていないことを示す。

【００５７】図１５は、認識サーバ２０４の「ｓｉｎｇｌｅ　ｍｏｄｅ」動作を示す絵画図である。「ａｕｔｏｍａｔｉｃ　ｍｏｄｅ」に関して上記で述べた属性およびイベントが適用可能であり、その場合は同じ参照番号で示されている。ただし、この動作モードでは、ｓｔｏｐ（）コール３０５が時間線２８１上に示されている。ｓｔｏｐ（）コール３０５は、ユーザによる「ｐｅｎ－ｕｐ」などのイベントに対応するものである。この動作モードでは、認識結果を戻すことが、明示的なｓｔｏｐ（）コール３０５の制御下にある。すべての動作モードと同様に、音声が「ｉｎｉｔｉａｌＴｉｍｅｏｕｔ」時間枠２８９内に検出されない場合は「ｏｎＳｉｌｅｎｃｅ」イベント２９１が発行されるが、この動作モードでは、認識は停止されない。同様に、ｓｔｏｐ（）コール３０５以前の認識できない発話によって生成された「ｏｎＮｏＲｅｃｏ」イベント２９３は、認識を停止しない。ただし、「ｂａｂｂｌｅＴｉｍｅｏｕｔ」属性２９５または「ｍａｘＴｉｍｅｏｕｔ」属性２９９に関連付けられた時間枠を超えると、認識は停止する。

【００５８】図１６は、認識サーバ２０４の「ｍｕｌｔｉｐｌｅ　ｍｏｄｅ」動作を示す絵画図である。前述のように、この動作モードは「ｏｐｅｎ－ｍｉｃｒｏｐｈｏｎｅ」の場合または口述シナリオで使用される。一般にこの動作モードでは、明示的なｓｔｏｐ（）＿コール３０５が受け取られるか、または「ｂａｂｂｌｅＴｉｍｅｏｕｔ」属性２９５または「ｍａｘＴｉｍｅｏｕｔ」属性２９９に関連付けられた時間枠を超えるまで、時々認識結果が戻される。ただし、認識を停止しない任意の「ｏｎＳｉｌｅｎｃｅ」イベント２９１、「ｏｎＲｅｃｏ」イベント３０３、または「ｏｎＮｏＲｅｃｏ」イベント２９３の後、「ｂａｂｂｌｅＴｉｍｅｏｕｔ」時間枠および「ｍａｘＴｉｍｅｏｕｔ」時間枠用のタイマはリセットされることに留意されたい。

【００５９】一般に、この動作モードでは、認識された各句について「ｏｎＲｅｃｏ」イベント３０３が発行され、ｓｔｏｐ（）コール３０５が受け取られるまで結果が戻される。認識できない発話により「ｏｎＳｉｌｅｎｃｅ」イベント２９１が発行された場合、これらのイベントは報告されるが、認識は続行する。

【００６０】前述のように、フィールド用の関連付けられたｒｅｃｏオブジェクトが起動され、これには少なくとも、文法を使用する認識サーバ２０４の指示を提供することが含まれる。この情報は、クライアント３０で記録され、認識サーバ２０４に送られる音声データを伴う

21

ことができる。前述のように、音声データは、ユーザが入力した音声に関連付けられたストリーミングデータを含むか、または音声認識時に使用される音声特徴を示す事前処理された音声データを含むことができる。他の実施形態では、認識サーバ２０４によって受け取られる音声データが、クライアント間で比較的一貫しているように、クライアント側の処理に音声データの正規化も含めることができる。これによって、認識サーバ２０４の音声処理が単純化され、認識サーバがクライアントおよび通信チャネルのタイプに支配されないようにできるため、認識サーバ２０４のスケーラビリティがより簡単になる。

【００６１】認識サーバ２０４から認識結果を受け取ると、認識結果は対応するフィールドに関連付けられ、所望であれば、クライアント側の検証またはチェックが実行できる。現在クライアントによって提示されているコードに関連付けられたすべてのフィールドが完了すると、情報はアプリケーション処理のためにＷｅｂサーバ２０２に送られる。前述の内容から、Ｗｅｂサーバ２０２は認識に好適なコードまたはページ／スクリプトをクライアント３０に提供するが、認識サービスはＷｅｂサーバ２０２ではなくむしろ認識サーバ２０４によって実行されることが明らかであろう。ただし、本発明は、認識サーバ２０４がＷｅｂサーバ２０２と共に配置されるか、または認識サーバ２０４がクライアント３０の一部であるような実施を排除するものではない。言い換えれば、本明細書に示される拡張部は、たとえ認識サーバ２０４がＷｅｂサーバ２０２またはクライアント３０と組み合わされる場合であっても、拡張部がこれら構成要素間に単純かつ便利なインターフェースを提供するので有益である。

【００６２】図８に示された実施形態には図示していないが、ｒｅｃｏ制御には、適切な音声データを認識サーバ２０４に向けて送るための、リモート可聴オブジェクト（ＲＡＯ）を含むこともできる。ＲＡＯをプラグインオブジェクトにすることの利点は、音声インターフェースは異なる可能性が多いため、異なるそれぞれのデバイスまたはクライアントに対して異なるものにすることができることである。さらにリモート可聴オブジェクトは、同時に複数のｒｅｃｏ要素を起動させることもできる。

【００６３】図９および図１０は、本明細書で実施される音声専用マークアップ言語を、ページ／スクリプトを備えたＨＴＭＬとして示す図である。図から明らかなように、コードには、本文部分３００およびスクリプト部分３０２も含まれている。ここにはマークアップ言語の他の拡張部、すなわちｂａｒｇｅｉｎなどの属性を含むプロンプト制御３０３がある。ただし、音声認識は、図９および図１０の音声専用実施形態では異なって行われる。ここではプロセスは、空のフィールドを特定し、対

22

応するプロンプトおよび新しいオブジェクトを起動する、スクリプト関数「ｃｈｅｃｋＦｉｌｌｅｄ」によって全体的に制御される。しかし、文法は、図８に関して上記で述べたものと同じ文脈を使用して起動され、音声データおよび使用する文法の指示が認識サーバ２０４に提供される。同様に、認識サーバ２０４から受け取った出力は、クライアントのフィールドに関連付けられる（本明細書では電話音声ブラウザ２１２）。

【００６４】一般に音声専用アプリケーションに固有の他の特徴は、音声が認識されなかったときのユーザに対する指示である。図８などのマルチモーダルアプリケーションでは、「ｏｎＮｏＲｅｃｏ」は、認識していないことを示す際に表示されたフィールドにｎｕｌｌ値を入れるだけであるため、それ以上の動作は必要ない。音声専用の実施形態では、「ｏｎＮｏＲｅｃｏ」３０５が関数「ｍｕｍｂｌｅ」を呼び出すかまたは実行し、語句を認識サーバ２０４に転送した後、これが好適なテキストから音声への変換システム３０７（図５）を使用して、音声に変換される。認識サーバ２０４は可聴ストリームを電話音声ブラウザ２１２に戻し、次にこれが電話８０に伝送されてユーザが聞くことになる。同様に、音声専用アプリケーションで実施された他の波形プロンプトも、必要なときに認識サーバ２０４によって可聴ストリームに変換される。

【００６５】この例では、関数「ｗｅｌｃｏｍｅ」を介してウェルカムプロンプトを再生した後、関数「ｃｈｅｃｋＦｉｌｌｅｄ」がユーザに対して各フィールドについてのプロンプトを出し、入力されたフィールドの反復および情報が正しいことの確認を含む適切な文法を起動するものであって、これには「ｃｏｎｆｉｒｍａｔｉｏｎ」文法の起動が含まれることに留意されたい。この実施形態では、それぞれのｒｅｃｏ制御が、以前の例の本文部分ではなくスクリプト部分３０２から開始されることに留意されたい。

【００６６】本発明の他の態様として、異なるタイプのクライアントデバイス（たとえば電話などの、マルチモーダルおよびディスプレイなし、音声入力ベースのクライアントデバイス）で実行可能なマークアップ言語は、それぞれのクライアントデバイスと対話するＷｅｂサーバについて、音声関連イベント、ＧＵＩイベント、および電話イベントのうち少なくとも１つを統合する。これは特に、Ｗｅｂサーバアプリケーションの重要な部分を包括的に、またはクライアントデバイスのタイプに関係なく書き込むことができるので有利である。図８および図９、図１０に、関数「ｈａｎｄｌｅ」を使用して一例が示されている。

【００６７】図９には示されていないが、マークアップ言語には、ＤＴＭＦ（Ｄｕａｌ　Ｔｏｎｅ　Ｍｏｄｕｌａｔｅｄ　Ｆｒｅｑｕｅｎｃｙ）制御および呼制御要素またはオブジェクトという、電話機能をサポートするた

23

めの拡張部がさらに２つある。ＤＴＭＦはｒｅｃｏ制御と同様の働きをする。キーパッド文字列をテキスト入力にマッピングする単一の文法を指定する。たとえば、「１」は食料雑貨部門、「２」は薬局部門を意味するなどである。これに対して、コールオブジェクトは、呼転送および第三者呼出しなどの電話機能を処理する。属性、プロパティ、メソッド、およびイベントについては、付録で詳細に論じる。

【００６８】図１１および図１２は、音声専用動作モードに好適な他のマークアップ言語例を示す図である。この実施形態では、ユーザは情報が入力されたかまたは話されたときに何らかの制御を行うことができる。言い換えれば、システムはユーザの話を開始させるか、そうでなければユーザに話を開始するように指示することが可能であるが、ユーザは最初に要求された以上の情報を提供することができる。これが「主導権混合（ｍｉｘｅｄ　ｉｎｉｔｉａｔｉｖｅ）」の一例である。一般にこの形態の対話（ｄｉａｌｏｇ　ｉｎｔｅｒａｃｔｉｏｎ）では、ユーザが対話の主導権をシステムと共有することができる。ユーザがプロンプトで要求された以上の情報を提供する、前述および以下で詳細に論じる例に加えて、ユーザは指示されていない場合にもタスクを切り換えることができる。

【００６９】図１１および図１２の例では、「ｄｏ＿ｆｉｅｌｄ」として示された文法に、文法「ｇ＿ｃａｒｄ＿ｔｙｐｅｓ」、「ｇ＿ｃａｒｄ＿ｎｕｍ」、および「ｇ＿ｅｘｐｉｒｙ＿ｄａｔｅ」に関連付けられた情報が含まれる。この例では、「ｏｎＲｅｃｏ」で示されるように認識された音声を受け取ると、電話音声ブラウザ２１２が電話８０から受け取った音声データおよび文法「ｄｏ＿ｆｉｅｌｄ」を使用するための指示を認識サーバ２０４に送信し、音声データからの認識された任意またはすべてのフィールドに関する値を関連付けることを含む関数「ｈａｎｄｌｅ」が、呼び出されるかまたは実行される。言い換えれば、認識サーバ２０４から取得した結果には、各フィールドに関する指示も含まれるということである。この情報が解析され、４０５に指定されたｂｉｎｄ規則に従って、対応するフィールドに関連付けられる。図５に示されるように、認識サーバ２０４は構文解析装置３０９を含むことができる。

【００７０】図７、図８、図９、図１０、図１１、および図１２では、非常に似通ったＷｅｂ開発フレームワークが使用されている。それぞれの場合のデータ提示も非常に似通っている。さらに、データ提示とフロー制御を分離することで、異なるアプリケーション間（システム主導および主導権混合）または異なるモダリティ（ｍｏｄａｌｉｔｙ）間（ＧＵＩ－Ｗｅｂベース、音声専用、およびマルチモーダル）で、最大の再利用性を可能にしている。さらにこれによって、電話がデバイス３０と同様のディスプレイおよび機能を含む場合に、音声専用動

24

作から電話を介してマルチモーダル動作へという、自然な拡張も可能にしている。付録Ａでは、上記で述べた制御およびオブジェクトについて、さらに詳細に説明する。

【００７１】上記のように、アップレベルブラウザは、認識結果を割り当てるために、上記の例で関数「ｈａｎｄｌｅ」を呼び出すことを含む様々な必要事項を実行するために、スクリプティングを使用することができる。前述の実施形態および付録Ａの２．１．２でさらに述べるように、「ｂｉｎｄ」要素は認識結果を解析して値を割り当てるものであって、「ｂｉｎｄ」要素は「ｒｅｃｏ」要素のサブ要素または子要素である。

【００７２】スクリプティングは有用であるが、たとえばセキュリティの問題により、これが最善のブラウザ実施形態ではないと考えている人が多い。したがって、本発明の他の実施形態または態様では、「ｂｉｎｄ」要素は高位要素（「ｒｅｃｏ」と同様）であって、他のより豊富なプロパティを備えており、本質的にはスクリプティングせずに、実質上はスクリプティングを模倣することができる。

【００７３】スクリプティングを使用しないか、または以下で論じる本発明の態様を使用しない、高度な対話効果などの以下で論じる機能の中には、ページをＷｅｂサーバ２０２に返すこと、新しいページを生成するためにアプリケーション論理を実行すること、およびページをクライアントデバイスに返信することによってのみ達成できるものもある。本発明のこの態様により、プログラマは、サーバのラウンドトリップなしにページのオブジェクトに関するメソッドを呼び出すことができる。

【００７４】前述の実施形態では、「ｂｉｎｄ」要素は、認識結果を書式またはＷｅｂページ内のフィールドに割り当てるための属性「ＴａｒｇｅｔＥｌｅｍｅｎｔ」および「ＴａｒｇｅｔＡｔｔｒｉｂｕｔｅ」のみを有する。他の実施形態では、「ｂｉｎｄ」要素は「ＴａｒｇｅｔＭｅｔｈｏｄ」も含んでおり、これがオブジェクトメソッド呼出し用に追加される。「ＴａｒｇｅｔＭｅｔｈｏｄ」の使用および機能は、スクリプティングを模倣する上で主要な技法である。たとえば以下の構文を使用して、オブジェクト「ＯＢＪ１」の「Ｘ」メソッドを呼び出すことができる。

<bind TargetElement = "OBJ1" TargetMethod = "X"...
>

【００７５】本明細書で示される例はＨＴＭＬ／ＸＨＴＭＬイベント構文に従うものであるが、当分野の技術者であれば、Ｗ３Ｃ　Ｄｏｃｕｍｅｎｔ　Ｏｂｊｅｃｔ　Ｍｏｄｅｌ　Ｌｅｖｅｌ　２またはＬｅｖｅｌ　３イベンティング標準、ＥＣＭＡＣｏｍｍｏｎ　ｌａｎｇｕａｇｅ　Ｉｎｆｒａｓｔｒｕｃｔｕｒｅ（ＣＬＩ）イベントモデル、Ｊａｖａ（登録商標）プログラミング言語イベントモデル、Ｗ３Ｃ　Ｓｙｎｃｈｒｏｎｏｕｓ　Ｍｕ

25

ｌｔｉｍｅｄｉａ　ＩｎｔｅｇｒａｔｉｏｎＬａｎｇｕａｇｅ（ＳＭＩＬ）、および近日発表されるＷ３Ｃ　ＸＭＬイベント標準提案を含むが、これらに限定されるものではない、他のイベンティング機構に、＜ｂｉｎｄ＞の用法を一般化するのは簡単であることを理解されることに留意されたい。

【００７６】図１７および図１８は、クライアント、具体的にはｄｏｗｎｌｅｖｅｌ　ｂｒｏｗｓｅｒ上で実行可能なマークアップ言語のページである。この例では、ユーザは可聴プロンプトを介して飲み物についてたずねられる。次にシステムは、何の飲み物が注文されたかを確認する。「ｂｉｎｄ」要素はこの認識結果に応じて、宣言論理を使用して実行を指導する。飲み物が確認されると、この書式はすべてのスクリプティングなしでＷｅｂサーバ２０２に返される。

【００７７】一般に、図１７および図１８のマークアップ例には、データセクション３５０、音声セクション３５２、およびユーザインターフェースセクション３５４、３５６、および３５８が含まれる。セクション３５４は、ユーザの希望する飲み物に関する一般問合せから認識結果を受け取り、クリームまたは砂糖が必要であるかどうかに関する再プロンプト、問合せ、あるいは注文された飲み物の確認のいずれかに向けて、対話型認識フローを送る。具体的に言えば、セクション３５６は、クリームまたは砂糖も注文されたときに認識結果を受け取る。セクション３５８は、飲み物の確認に関する認識結果を受け取る。セクション３６０は新しいメッセージングオブジェクト「ＳＭＥＸ」を使用する呼制御セクションであり、これについては以下で詳細に論じる。

【００７８】前述のように、本発明のこの態様の「ｂｉｎｄ」要素にはオブジェクトメソッド呼出しが含まれ、これが３６１で「ｗｅｌｃｏｍｅ」オブジェクトの「ｓｔａｒｔ」メソッドが実行されたときに、「ｗｅｌｃｏｍｅ」プロンプトを再生することによって、図１７および図１８の例でユーザ対話を開始させる。

【００７９】次にユーザは、３６２で「ａｓｋｅｄ」オブジェクトの「ｓｔａｒｔ」メソッドを実行することによって、「Ｄｏ　ｙｏｕ　ｗａｎｔ　ｃｏｋｅ，ｃｏｆｆｅｅ　ｏｒ　ｏｒａｎｇｅ　ｊｕｉｃｅ？」とたずねられる。次いで、３６３で「ｒｅｃｏ＿ｄｒｉｎｋ」オブジェクトの「ｓｔａｒｔ」メソッドを呼び出すことによって、認識が実行される。

【００８０】次にセクション３５４のマークアップが実行され、認識サーバ２０４が使用する文法が、Ｘｐａｔｈステートメント「．／ｄｒｉｎｋ　ｔｙｐｅｓ」によって提供される。この例では、Ｗ３ＣのＸｐａｔｈ言語を使用しているが、当分野の技術者であれば、Ｗ３ＣのＸＭＬ照会言語（ＸＱＬ）を含むがこれに限定されることのない他の標準言語にまで概念を広げることが簡単であることを理解することに留意されたい。「ｂｉｎｄ」

26

要素３６４で指定されるように、認識サーバ２０４から受け取った認識結果の確信値が１０よりも小さい場合、３６６でプロンプトオブジェクト「ｒｅｐｒｏｍｐｔ」が、続いて３６８でプロンプトオブジェクト「ａｓｋ」が実行され、３７０で認識オブジェクト「ｒｅｃｏ＿ｄｒｉｎｋ」が再開される。戻された認識結果が、確信値が１０よりも大きい「ｃｏｆｆｅｅ」の場合、３７２でフィールド「ｄｒｉｎｋ」に認識結果の値が割り当てられ、次いでユーザは、３７４で、プロンプトオブジェクト「ｃｒｅａｍ＿ｓｕｇａｒ」によってクリームまたは砂糖を希望するか否かに関するプロンプトを受け取る。次いで３７６で、セクション３５６の認識オブジェクト「ｒｅｃｏ＿ｃｒｅａｍ＿ｓｕｇａｒ」が呼び出される。これとは違い、認識結果の確信値が１０よりも大きいがｃｏｆｆｅｅでない場合、３７８で再度ｄｒｉｎｋフィールドが割り当てられる。３８０でプロンプトオブジェクト「ｃｏｎｆｉｒｍ」を実行し、その後セクション３５８の３８２で認識オブジェクト「ｒｅｃｏ＿ｙｅｓｎｏ」を呼び出すことによって、認識結果の確認が与えられる。ユーザが１０より大きい確信値で「ｙｅｓ」と答えると、３８４でプロンプトオブジェクト「ｔｈａｎｋｓ」が再生され、その後３８６で書式がサブミットされる。これとは違い、ユーザが「ｎｏ」と答えるか、または認識結果の確信値が１０より小さい場合、３９０でプロンプトオブジェクト「ｒｅｔｒｙ」が実行され、もう一度３９２でプロンプトオブジェクト「ａｓｋ」が実行されて、３９４で認識オブジェクト「ｒｅｃｏ＿ｄｒｉｎｋ」が呼び出される。

【００８１】前述の例からわかるように、「ｂｉｎｄ」要素は、セクション３５４、３５６、または３５８に示すようにメソッドの複数呼出しを可能にする。所望であれば、認識された結果の複数割当てを宣言することもできる。例示された実施形態では、複数の割当ておよびメソッド呼出しが宣言されると、ドキュメント順で実行される。

【００８２】他の実施形態では、メソッド引数を渡すための規定も提供される。言い換えれば、メソッドの中には引数リストが必要なものもある。これは、「ａｒｇ」サブ要素を使用して達成される。たとえば、以下のマークアップ

<bind TargetElement = "OBJ" TargetMethod = "F"><arg>X</arg><arg>Y</arg></bind>

は、「ＯＢＪ．Ｆ（Ｘ，Ｙ）」と等価であるか、あるいは「ＯＢＪ」が、パラメータまたは引数「Ｘ」および「Ｙ」を使用するメソッド「Ｆ」を有するオブジェクトである。

【００８３】「ｂｉｎｄ」要素は、ｂｉｎｄ要素がどのイベント用であるかを宣言する「ｅｖｅｎｔ」属性を含むことも可能である。たとえば、以下のマークアップ

<bind event = "onNoReco" = TargetElement =

27

"prompt1" TargetMethod = "start"/>

は、「ｏｎＮｏＲｅｃｏ」イベントが送られたときを意味するものであり、オブジェクト「ｐｒｏｍｐｔ１」の「ｓｔａｒｔ」メソッドが呼び出されることになる。図８に関して上記の例で説明したように、「ｂｉｎｄ」要素を「Ｒｅｃｏ」要素の子要素として使用する場合と整合性を持たせるために、「ｂｉｎｄ」要素のデフォルト属性は「ｏｎＲｅｃｏ」である。

【００８４】高位要素としての「ｂｉｎｄ」要素は、付録の２．４項に指定されたいずれのイベントも含むことができる。さらに、「ｂｉｎｄ」要素は、アクセス可能であってプログラムフローの向きを決めるのに使用できる「ｓｔａｔｕｓ」属性を有する、「ｏｎＥｒｒｏｒ」イベントを含むこともできる。「ｂｉｎｄ」要素の他のイベントが「ｓｔａｔｕｓ」属性を有する場合、これらにも同様にアクセスできる。

【００８５】認識結果の条件をチェックすることに加えて、実行予定の現在のドキュメントまたはページもチェックすることができる。具体的に言えば、「ｔｅｓｔ」属性および「ｖａｌｕｅ」属性の両方を、格納ドキュメントのルートノードを示す「ｈｏｓｔ」プリミティブを含むように拡張することができる。たとえば、図１７、図１８を再度参照すると、そこに含まれる例は、セクション３５４に、ユーザがｃｏｆｆｅｅを要求したときにクリームまたは砂糖を希望するかどうかをたずねるための追加論理がある。クリームまたは砂糖を追加するため、したがって、マークアップ「ｈｏｓｔ（）／ｇｅｔ＿ｄｒｉｎｋ／ｄｒｉｎｋ＝「ｃｏｆｆｅｅ」」によって指定されるように、セクション３５６を呼び出すためのフラグは、ｄｒｉｎｋフィールドが「ｃｏｆｆｅｅ」の場合に限ってオンになる。

【００８６】また、「ｂｉｎｄ」要素は音声サーバ２０４からの認識結果およびドキュメント内での値の受取りまたは割当てに適用可能なだけでなく、たとえば、クライアントデバイス上で実行中のアプリケーションからのメッセージオブジェクト（本明細書では「ｓｍｅｘ」で示されている）にも適用可能であることに留意されたい。図１７および図１８の例では、クライアントデバイス上で実行中の電話アプリケーションが呼を検出したときに、ページが実行される。セクション３６０では、「ｂｉｎｄ」要素は、メッセージ「／Ｃａｌｌ＿ｃｏｎｎｅｃｔｅｄ」が受け取られたときに、「ｗｅｌｃｏｍｅ」プロンプトを実行または再生し、「ｒｅｃｏ＿ｄｒｉｎｋ」オブジェクトを実行することによって認識を開始する。音声サーバ２０４から受け取る認識結果と同様に、受け取るメッセージも多種多様である。メッセージの中には、所望のプログラムフローを開始するために首尾よく定義されたものもある。その他のものも受け取って処理する（たとえば、受け取った認識サーバの認識結果と同様に解析する）ことができる。たとえばこれによって、キーボードからのテキストの本来の言語構文解析装置と同様に、マークアップを使用することができる。付録Ａのｒｅｃｏ要素は、この機能を実行するためのプロパティを含む。同様に、付録Ａで詳細に説明するが、プロパティ「ｉｎｎｅｒｔｅｘｔ」を使用することによって、プロンプト要素を使用して動的コンテンツまたは可聴波ファイルにテキストメッセージを提供することができる。イベンティングは認識結果に関するイベンティングと同様とすることができる。たとえば、イベンティングは「ｏｎＲｅｃｅｉｖｅｄ」を含むことが可能であり、これは、メッセージソース（たとえばクライアントデバイス上で実行中のアプリケーション）がブラウザに使用可能なメッセージを有する場合に送られる。

【００８７】したがって「ｓｍｅｘ」またはメッセージオブジェクトは、本明細書で論じるマークアップタグを、クライアントデバイス上で実行中の他の構成要素またはアプリケーションまで拡張できるようにするものである。他の例として、メッセージオブジェクトは、クライアントデバイス上で実行中の耳の不自由な人向けＴＴＹ構成要素への通信に使用することができる。ＴＴＹ構成要素は、音声認識を使用する代わりに、ユーザが入力した内容のメッセージを提供する。その後このメッセージは、認識サーバから認識結果が受け取られた場合と同じように使用され、メッセージが解析されて書式のフィールドに割り当てられるか、または前述の「ｒｅｃｏ」、「ｇｒａｍｍａｒ」、または「ｂｉｎｄ」などの要素を使用して、他の処理を実行することができる。メッセージまたは「ｓｍｅｘ」オブジェクトについては、付録Ａで詳細に説明する。

【００８８】「ｂｉｎｄ」要素は、「ｆｏｒ」属性を含むことも可能であり、これによってその動作をページ上の他のオブジェクトに結びつけることができる。たとえば、以下のようなマークアップ

<bind for = "prompt 1" event = "onComplete" targetElement = "prompt 2" = targetMethod = "start" />

は、オブジェクト「ｐｒｏｍｐｔ　１」がイベント「ｏｎＣｏｍｐｌｅｔｅ」を送信したときに、オブジェクト「ｐｒｏｍｐｔ　２」のｓｔａｒｔメソッドを呼び出す。

【００８９】再度図５を参照すると、Ｗｅｂサーバ２０２はサーバ側のプラグイン宣言型オーサリングツールまたはモジュール３２０を含むことができる（たとえば、Ｍｉｃｒｏｓｏｆｔ　ＣｏｒｐｏｒａｔｉｏｎによるＡＳＰまたはＡＳＰ＋、ＪＳＰなど）。サーバ側プラグインモジュール３２０は、クライアント側マークアップ、ならびにＷｅｂサーバ２０２にアクセス中のクライアントのタイプに特有な書式のマークアップでさえも、動的に生成することが可能である。クライアント／サーバの関係が最初に確立された時点で、クライアント情報がＷ

29

ｅｂサーバ２０２に提供されるか、またはＷｅｂサーバ２０２が、クライアントの機能を検出するためのモジュールまたはルーチンを含むことができる。この方式では、サーバ側プラグインモジュール３２０が、各音声認識シナリオについて、すなわち電話８０を介した音声専用の、またはデバイス３０向けマルチモーダルの、クライアント側マークアップを生成することができる。一貫したクライアント側モデル（各アプリケーションで使用できるｒｅｃｏおよびプロンプト制御）を使用することによって、多くの異なるクライアント向けにアプリケーションをオーサリングすることが非常に容易になる。

【００９０】クライアント側マークアップを動的に生成することに加えて、図８、図９、および図１０のマークアップ例を使用して図６に図示したクレジットカード情報を取得するなどの高位ダイアログモジュールは、アプリケーションをオーサリング中の開発者が使用するために、サーバ側制御が記憶装置３２４に格納される場合と同様に実施することができる。一般に、高位ダイアログモジュール３２４は、開発者によって指定されたパラメータに基づいて、クライアント側マークアップおよびスクリプトを、音声専用およびマルチモーダルの両方のシナリオで動的に生成する。高位ダイアログモジュールは、開発者のニーズに合うようなクライアント側マークアップを生成するためのパラメータを含むことができる。たとえば、クレジットカード情報モジュールは、クライアント側マークアップスクリプトがどのタイプのクレジットカードを使用できるかを示すパラメータを含むことができる。サーバ側プラグインモジュール３２０で使用するＡＳＰ＋ページの例が、図１３に示されている。

【００９１】以上、本発明について、好ましい実施形態を参照しながら説明してきたが、当分野の技術者であれば、本発明の精神および範囲を逸脱することなく、形態および詳細の変更が可能であることを理解されよう。

【００９２】付録Ａ

１　はじめに

以下のタグは、ドキュメントが音声を入力または出力媒体として使用できるようにする、マークアップ要素のセットである。タグは、ＨＴＭＬ、ＸＨＴＭＬ、ｃＨＴＭＬ、ＳＭＩＬ、ＷＭＬ、などのような任意のＳＧＭＬ導出型マークアップ言語に埋め込むことのできる、独立型ＸＭＬとなるように設計される。本明細書で使用されるタグは、ワシントン州ＲｅｄｍｏｎｄのＭｉｃｒｏｓｏｆｔ　Ｃｏｒｐｏｒａｔｉｏｎから市販されているよく知られたメソッドである、ＳＡＰＩ　５．０と同様である。タグ、要素、イベント、属性、プロパティ、戻り値などは、単なる例示的なものであって、限定的なものとみなしてはならない。本明細書では音声およびＤＴＭＦ認識について例示しているが、同様のタグを他の認識形態にも提供することができる。

30

【００９３】本明細書で論じた主な要素は、以下の通りである。

＜ｐｒｏｍｐｔ．．．＞　音声合成構成およびプロンプトを再生する場合

＜ｒｅｃｏ．．．＞　認識装置構成ならびに認識の実行および後処理をする場合

＜ｇｒａｍｍａｒ．．．＞　入力文法リソースを指定する場合

＜ｂｉｎｄ．．．＞　認識結果を処理する場合

＜ｄｔｍｆ．．．＞　ＤＴＭＦを構成および制御する場合

【００９４】２　Ｒｅｃｏ

Ｒｅｃｏ要素は、可能なユーザ入力および入力結果を処理するための手段を指定する際に使用する。したがって、主な要素は＜ｇｒａｍｍａｒ＞および＜ｂｉｎｄ＞であり、認識装置のプロパティを構成するためのリソースを含む。

【００９５】Ｒｅｃｏ要素は、ＳｔａｒｔおよびＳｔｏｐメソッドを介してアップレベルブラウザで、あるいはＳＭＩＬコマンドを使用してＳＭＩＬで使用可能なブラウザで、プログラムに基づいて活動化される。ページ上に存在することで、ダウンレベルブラウザ（すなわち、非スクリプトサポーティングブラウザ）では宣言的に活動状態であるとみなされる。複数の文法を並行して活動化できるようにするために、複数のＲｅｃｏ要素を同時に活動状態であるとみなすことができる。

【００９６】Ｒｅｃｏは、実行できる認識シナリオの種類および認識プラットフォームの動作を区別するために、特定のモード、すなわち「ａｕｔｏｍａｔｉｃ」、「ｓｉｎｇｌｅ」、または「ｍｕｌｔｉｐｌｅ」を採用することも可能である。

【００９７】２．１　Ｒｅｃｏコンテンツ

Ｒｅｃｏ要素には、１つまたは複数の文法および任意選択でｂｉｎｄ要素のセットが含まれ、これが認証結果を検査し、値に関連する部分を含む側のページにコピーする。

【００９８】アップレベルブラウザでは、Ｒｅｃｏが個々の文法規則のプログラムに基づく活動化および非活動化をサポートする。認識文脈に関しては、デフォルト時に文法中のすべての最高位規則が活動状態であることにも留意されたい。

【００９９】２．１．１　＜ｇｒａｍｍａｒ＞要素

ｇｒａｍｍａｒ要素は、ｓｒｃ属性を使用するインラインまたは参照済みどちらかの文法を指定するのに使用される。典型的には、少なくとも１つの文法（インラインまたは参照済み）が指定される。インライン文法は、テキストベースの文法フォーマットであってよいが、参照済み文法はテキストベースまたはバイナリタイプであってよい。複数のｇｒａｍｍａｒ要素を指定することができる。複数のｇｒａｍｍａｒ要素が指定される場合、文

31

法内の規則は、同じ文法内の特別規則として追加される。同じ名前を持つどんな規則も上書きされる。

【０１００】属性：

・ｓｒｃ：インライン文法が指定される場合、任意選択である。文法のＵＲＩが含まれる。文法内のすべての最高位規則は、認識文脈に関してデフォルト時に活動状態であることに留意されたい。

・ｌａｎｇＩＤ：任意選択である。どの言語音声エンジンを使用するべきであるかを示す文字列。文字列フォーマットはｘｍｌ：ｌａｎｇ定義に従う。たとえば、ｌａｎｇＩＤ＝「ｅｎ－ｕｓ」は米国英語を示す。この属性は、ｌａｎｇＩＤが文法ＵＲＩに指定されていない場合に限って有効である。指定されていない場合、デフォルトの値は米国英語である。

【０１０１】ｌａｎｇＩＤが複数箇所で指定される場合、ｌａｎｇＩＤは最低位範囲から優先順位に従うものであり、すなわちリモート文法ファイル（すなわち、言語ｉｄが文法範囲内で指定される）の次にｇｒａｍｍａｒ要素、その次にｒｅｃｏ要素となる。

```
〈grammar src="FromCity.xml" /〉
    or
〈grammar〉
    〈rule toplevel="active"〉
        〈p〉from 〈/p〉
        〈ruleref name="cities" /〉
    〈/rule〉
    〈rule name="cities"〉
        〈1〉
        〈p〉 ケンブリッジ 〈/p〉
        〈p〉 シアトル 〈/p〉
        〈p〉 ロンドン 〈/p〉
        〈/1〉
    〈/rule〉
〈/grammar〉
```

【０１０２】ｓｒｃ参照済み文法およびインライン文法＊

```
<sml confidence="40">
    <travel text="シアトルからボストンに行きたい">
        <origin_city confidence="45"> シアトル
</origin_city>
            <dest_city  confidence="35"> ボストン
</dest_city>
        </travel>
</sml>
```

【０１０５】文法での認識は、意味マークアップ言語またはＳＭＬでＸＭＬドキュメントを作成することが想定されているため、ＳＭＬドキュメントからｂｉｎｄされる値はＸＰａｔｈ照会を使用して参照される。また、値のｂｉｎｄ先となるページ内の要素は一意に識別されるべきである（形態制御となる見込みである）ため、これらのターゲット要素は直接参照される。

【０１０６】属性：

・ｔａｒｇｅｔＥｌｅｍｅｎｔ：必須。ＳＭＬからのｖａｌｕｅコンテンツが割り当てられる要素（Ｗ３Ｃ　ＳＭＩＬ　２．０の場合と同様）。

・ｔａｒｇｅｔＡｔｔｒｉｂｕｔｅ：任意選択。ＳＭＬからのｖａｌｕｅコンテンツが割り当てられるターゲット要素の属性（ＳＭＩＬ　２．０のａｔｔｒｉｂｕｔｅ

32

＊の両方が指定された場合、インライン規則が参照済み規則に追加され、同じ名前を有するどんな規則も上書きされる。

【０１０３】２．１．２　＜ｂｉｎｄ＞要素

ｂｉｎｄ要素は認識結果からの値をページに結合するのに使用される。

【０１０４】ｂｉｎｄ要素によって消費される認識結果は、認識結果を指定するための意味マークアップ言語（ＳＭＬ）を含むＸＭＬドキュメントであってよい。そのコンテンツには、意味値、実際に話された言葉、および確信値が含まれる。ＳＭＬは代替の認識選択肢をふくむこともできる（Ｎ最高認識結果の場合と同様）。「シアトルからボストンに行きたい」という発話に対するＳＭＬドキュメントの例を以下に示す。

33

Ｎａｍｅ属性を使用する場合と同様）。指定されない場合、デフォルトの値は「ｖａｌｕｅ」。

・ｔｅｓｔ：任意選択。認識結果が割り当てられる条件を示すXML Pattern（W3C XML DOM指定の場合と同様）文字列。デフォルト条件はｔｒｕｅ。

・ｖａｌｕｅ：必須。ターゲット要素に割り当てられる認識結果ドキュメントからの値を指定する、XPATH（W3C XML DOM指定の場合と同様）文字列。

【０１０７】例：上記SMLの戻りが与えられると、以下のｒｅｃｏ要素は、ｏｒｉｇｉｎ＿ｃｉｔｙおよびｄｅｓｔ＿ｃｉｔｙ内の値を、ターゲットページ要素ｔｘｔＢｏｘＯｒｉｇｉｎおよびｔｘｔＢｏｘＤｅｓｔに転送するためにｂｉｎｄを使用する。

```
<input name="txtBoxOrigin" type="text"/>
<input name="txtBoxDest" type="text" />

<reco id="travel">
    <grammar src="./city.xml" />
    <bind targetElement="txtBoxOrigin"
        value="//origin_city" />
    <bind targetElement="txtBoxDest"
        value="//dest_city" />
</reco>
```

【０１０８】このｂｉｎｄは、以下の例に示されるよう＊に、条件付きとすることが可能であり、ｂｉｎｄ動作の前提条件として、ｄｅｓｔ＿ｃｉｔｙ結果の確信属性に関するテストが実行される。

```
<bind targetElement="txtBoxDest"
    value="//dest_city"
    test="/sml/dest_city[@confidence $gt$ 40]"
/>
```

【０１０９】ｂｉｎｄ要素は、ｄｏｗｎｌｅｖｅｌまたはアップレベルブラウザで認識結果を処理する単純な宣言手段である。より複雑な処理の場合、アップレベルブラウザがサポートするｒｅｃｏ DOMオブジェクトは、プログラムに基づくスクリプト分析および認識戻り値の後処理を可能にするために、ｏｎＲｅｃｏイベントハンドラを実施する。

【０１１０】２．２ 属性およびプロパティ

以下の属性は、すべてのブラウザによってサポートされ、プロパティはアップレベルブラウザによってサポートされる。

【０１１１】２．２．１ 属性

以下のＲｅｃｏの属性は、ダイアログターン（ｄｉａｌｏｇ ｔｕｒｎ）に関して音声認識装置を構成するのに使用される。

・ｉｎｉｔｉａｌＴｉｍｅｏｕｔ：任意選択。認識開始から音声検出までのミリ秒単位の時間。この値が認識プラットフォームに渡され、これを超えると、認識プラットフォームからｏｎＳｉｌｅｎｃｅイベントが提供される（２．４．２を参照）。指定されない場合、音声プラットフォームはデフォルト値を使用する。

・ｂａｂｂｌｅＴｉｍｅｏｕｔ：任意選択。認識装置が音声検出後に結果を戻さなければならないミリ秒単位の時間枠。ａｕｔｏｍａｔｉｃおよびｓｉｎｇｌｅモードでのｒｅｃｏの場合、これは音声検出から停止コールまでの時間枠に適用される。「ｍｕｌｔｉｐｌｅ」モードでのｒｅｃｏの場合、このタイムアウトは音声検出から各認識戻りまでの時間枠に適用され、すなわちこの時間枠は、それぞれの結果が戻されるかまたは他のイベント後に再開される。これを超えると、エラーが発生したか否かに従って、様々なイベントが投じられる。認識装置が依然として音声を処理している場合、たとえば非常に長い発話の場合などは、ｓｔａｔｕｓコード１３（２．４．４を参照）でｏｎＮｏＲｅｃｏイベントが投じられる。ただし他のなんらかの理由でタイムアウトを超えた場合は、認識装置のエラーである可能性が高いため、ｏｎＴｉｍｅｏｕｔイベントが投じられる。指定されない場合は、音声プラットフォームはデフォルトとして内部値をとる。

・ｍａｘＴｉｍｅｏｕｔ：任意選択。認識開始から結果がブラウザに戻されるまでの、ミリ秒単位の時間枠。これを超えると、ブラウザによってｏｎＴｉｍｅｏｕｔイベントが投じられ、これは、分散環境でのネットワークまたは認識装置の障害に応じるものである。「ｍｕｌｔｉｐｌｅ」モードでのｒｅｃｏの場合、ｂａｂｂｌｅＴｉｍｅｏｕｔを使用する場合と同様に、この時間枠は、それぞれの結果が戻されるかまたは他のイベント後に再開される。ｍａｘＴｉｍｅｏｕｔ属性は、ｉｎｉｔｉａｌＴｉｍｅｏｕｔおよびｂａｂｂｌｅＴｉｍｅｏｕｔの合計より大きいかまたは等しくなければならないことに留意されたい。指定されない場合は、値はブラウザのデフォルト値となる。

・ｅｎｄＳｉｌｅｎｃｅ：任意選択。ａｕｔｏｍａｔｉｃモードでのＲｅｃｏの場合、その後認識結果が戻される、音声がないはずの発話終了後の無音の、ミリ秒単位の時間枠。ｒｅｃｏの場合はａｕｔｏｍａｔｉｃ以外のモードは無視される。指定されない場合は、デフォルトとしてプラットフォーム内部値をとる。

・ｒｅｊｅｃｔ：任意選択。認識拒否のしきい値であって、これを下回るとプラットフォームは「ｎｏ ｒｅｃｏ」イベントを投じる。指定されない場合、音声プラットフォームはデフォルト値を使用する。確信値は０から１００まで（整数）。拒否値はこの間となる。

・ｓｅｒｖｅｒ：任意選択。音声プラットフォームのＵＲＩ（タグインタープリタと認識プラットフォームが共に配置されていない場合に使用する）。値の一例は、ｓｅｒｖｅｒ＝ｐｒｏｔｏｃｏｌ：／／ｙｏｕｒｓｐｅｅｃｈｐｌａｔｆｏｒｍである。アプリケーション作者は、たとえば次のようにＵＲＩ文字列に照会文字列を追加することによって、音声プラットフォーム特有の設定を提供することもできる。

protocol://yourspeechplatform?bargeinEnergyThreshold=0.5

・ｌａｎｇＩＤ：任意選択。どの言語音声エンジンを使用するべきかを示す文字列。文字列フォーマットは、ｘｍｌ：ｌａｎｇ定義に従う。たとえば、ｌａｎｇＩＤ＝「ｅｎ－ｕｓ」は、米国英語を示す。この属性は、ｇｒａｍｍａｒ要素にｌａｎｇＩＤが指定されない場合に限って有効である（２．１．１を参照）。

・ｍｏｄｅ：任意選択。追従する認識モードを指定する文字列。指定されない場合は、デフォルトとして「ａｕｔｏｍａｔｉｃ」モードをとる。

【０１１２】２．２．２ プロパティ

以下のプロパティには、認識プロセスによって戻される結果が含まれる（アップレベルブラウザによってサポートされる）。

・ｒｅｃｏＲｅｓｕｌｔ：読取り専用。２．１．２で記載したように、意味マークアップ言語（SML）を含むXML DOMノードオブジェクト内に保持される認識の結果。何も認識されない場合、プロパティはｎｕｌｌを戻す。

・ｔｅｘｔ：読取り／書込み。認識された言葉のテキストを保持する文字列（すなわち、読取りモードでのｒｅｃｏＲｅｓｕｌｔで、SML認識戻りでの最高位要素のテキスト属性のコンテンツに関する略記）。書込みモードの場合、文字列を割り当てることが可能であり、次いでこれが、文字列が認識結果に対応するかのように解析される。書込みモードは、マークアップタグの拡張およびその処理を、クライアントデバイス上の他の構成要素またはアプリケーションに対しても可能にするものである。文字列は「ｓｍｅｘ」メッセージオブジェクトから取得することができる。

・ｓｔａｔｕｓ：読取り専用。認識プラットフォームから戻される状態コード。可能な値は、認識が成功した場合は０、または失敗値は－１から－４（Ｓｔａｒｔメソッド（セクション２．３．１）およびＡｃｔｉｖａｔｅメソッド（セクション２．３．４）での可能な例外に定義されたとおり）、ならびに認識装置イベントの受取り時に設定される状態値－１１から－１５（２．４を参照）である。

【０１１３】２．３ オブジェクトメソッド

Ｒｅｃｏの活動化およびｇｒａｍｍａｒの活動化は、ＲｅｃｏのDOMオブジェクトにあるいかのメソッドを使用して制御することができる。これらのメソッドを使用すると、アップレベルブラウザがＲｅｃｏオブジェクトを開始および停止すること、進行中の認識を取り消すこと、ならびに個々の文法最高位規則（アップレベルブラウザのみ）を活動化および非活動化することができる。

【０１１４】２．３．１ Ｓｔａｒｔ

Ｓｔａｒｔメソッドは、明示的に非活動化されていない認識文脈に対してすべての最高位規則を活動状態の文法として使用することで、認識プロセスを開始する。

【０１１５】構文：

Ｏｂｊｅｃｔ．Ｓｔａｒｔ ()

戻り値：

なし

例外：このメソッドは、非ゼロのｓｔａｔｕｓコードを設定し、障害時にはｏｎＮｏＲｅｃｏイベントを開始（ｆｉｒｅ）する。可能な障害には、文法がない（ｒｅｃｏ ｓｔａｔｕｓ＝－１）、文法のコンパイル障害と同様に様々な理由で発生することのある文法のロード障害、ＵＲＩが存在しない（ｒｅｃｏ ｓｔａｔｕｓ＝－２）、または音声プラットフォームのエラー（ｒｅｃｏ ｓｔａｔｕｓ＝－３）が含まれる。

【０１１６】２．３．２ Ｓｔｏｐ

Ｓｔｏｐメソッドは、認識プロセスを終了させるためのコールである。Ｒｅｃｏオブジェクトは音声の録音を停止し、認識装置は受け取った音声に関する認識結果を、録音が停止した時点まで戻す。Ｒｅｃｏが使用するすべての認識リソースが解除され、その文法が非活動化される。（このメソッドは、認識装置それ自体が、完全な文章を認識した後に検出終了時点でｒｅｃｏオブジェクト

を停止するので、ａｕｔｏｍａｔｉｃモードでの典型的な認識のために明示的に使用される必要のないことに留意されたい。）Ｒｅｃｏが開始されていない場合、コールは無効である。

【０１１７】構文：

Ｏｂｊｅｃｔ．Ｓｔｏｐ ()

戻り値：

なし

例外：

なし

【０１１８】２．３．３　Ｃａｎｃｅｌ

Ｃａｎｃｅｌメソッドは、音声を認識装置に送るのを停止し、文法を非活動化して認識装置を解除し、どのような認識結果も廃棄する。ブラウザは、取り消された認識に関する認識結果を無視する。認識装置が開始されていない場合、コールは無効である。

【０１１９】構文：

Ｏｂｊｅｃｔ．Ｃａｎｃｅｌ ()

戻り値：

なし

例外：

なし

【０１２０】２．３．４　Ａｃｔｉｖａｔｅ

Ａｃｔｉｖａｔｅメソッドは、文脈自由文法（ＣＦＧ）の最高位規則を活動化する。活動化は、「Ｓｔａｒｔｅｄ」認識プロセス中は無効となるため、認識が開始される前に呼び出さなければならない。明示的に非活動化されていない認識文脈に関するすべての文法最高位規則は、すでに活動状態として処理されていることに留意されたい。

【０１２１】構文：

Ｏｂｊｅｃｔ．Ａｃｔｉｖａｔｅ（ｓｔｒＮａｍｅ）；

パラメータ：

＊・ｓｔｒＮａｍｅ：必須。活動化される規則の名前

戻り値：

なし

例外：

なし

【０１２２】２．３．５　Ｄｅａｃｔｉｖａｔｅ

このメソッドは、文法の最高位規則を非活動化する。規則が存在しない場合、メソッドは無効である。

【０１２３】構文：

Ｏｂｊｅｃｔ．Ｄｅａｃｔｉｖａｔｅ（ｓｔｒＮａｍｅ）；

パラメータ：

・ｓｔｒＮａｍｅ：必須。非活動化される規則の名前。文字列が空であると、すべての規則が非活動化される。

戻り値：

なし

例外：

なし

【０１２４】２．４　Ｒｅｃｏイベント

Ｒｅｃｏ　ＤＯＭオブジェクトは以下のイベントをサポートするが、そのハンドラは、ｒｅｃｏ要素の属性として指定することができる。

【０１２５】２．４．１　ｏｎＲｅｃｏ：このイベントは、認識装置がブラウザに使用できる認識結果を得たときに開始される。ａｕｔｏｍａｔｉｃモードでのｒｅｃｏの場合、このイベントは認識プロセスを自動的に停止し、リソースを消去する（２．３．２を参照）。ｏｎＲｅｃｏは、典型的には、認識結果のプログラムに基づいた分析と、結果をページに入れる処理に使用される。

【０１２６】構文：

【０１２７】

【表１】

＊

インラインＨＴＭＬ	<Reco onReco="handler" >
イベントプロパティ	Object.onReco = handler; Object.onReco = GetRef("handler");

【０１２８】イベントオブジェクト情報：　※【表２】

【０１２９】

※

バブル	なし
呼出し方法	ユーザが何かを言う
デフォルト動作	認識結果オブジェクトを戻す

【０１３０】イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる（イベントオブジェクトの用法については以下の例を参照）。

【０１３１】例：以下のＸＨＴＭＬフラグメントは、ｏｎＲｅｃｏを使用して認識出力を解析するためのスクリプトを呼び出し、適切なフィールドに値を割り当てる。

```
<input name="txtBoxOrigin" type="text" />
<input name="txtBoxDest" type="text" />
<reco onReco="processCityRecognition()"/>
    <grammar src="/grammars/cities.xml" />
```



```
</reco>

<script><![CDATA[
    function processCityRecognition () {
        smlResult = event.srcElement.recoResult;
        origNode =
smlResult.selectSingleNode("//origin_city");
        if (origNode != null) txtBoxOrigin.value =
origNode.text;
        destNode =
smlResult.selectSingleNode("//dest_city");
        if (destNode != null) txtBoxDest.value =
destNode.text;
    }
]]></script>
```

【０１３２】２．４．２　ｏｎＳｉｌｅｎｃｅ：ｏｎＳｉｌｅｎｃｅは、ＲｅｃｏのｉｎｉｔｉａｌＴｉｍｅｏｕｔ属性で指定された持続期間（２．２．１を参照）の前に、認識プラットフォームによって検出された音声のないイベントを処理する。このイベントは、ａｕｔｏｍ＊ａｔｉｃｒｅｃｏｇｎｉｔｉｏｎモードの場合に認識プロセスを自動的に取り消す。

【０１３３】構文：

【０１３４】

【表３】

インラインＨＴＭＬ	<reco onSilence="handler" ...>
イベントプロパティ （ＥＣＭＡＳｃｒｉｐｔの場合）	Object.onSilence = handler Object.onSilence = GetRef("handler");

【０１３５】イベントオブジェクト情報：　※【表４】

【０１３６】

※

バブル	なし
呼出し方法	認識装置が、initialTimeout 属性で指定された時間枠内に、何の音声も検出しなかった。
デフォルト動作	status = -11 に設定

【０１３７】イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。

【０１３８】２．４．３　ｏｎＴｉｍｅｏｕｔ

ｏｎＴｉｍｅｏｕｔは、典型的には音声プラットフォームからのエラーを反映する、２種類のイベントを処理する。

・認識が完了する前に満了となったｍａｘｔｉｍｅ属性（２．２．１を参照）に指定された時間枠を送信する、タグインタープリタによって投じられたイベントを処理する。このイベントは、典型的には、分散アーキテクチ★★ャで発生する可能性のある問題を反映する。

・認識が開始されたにもかかわらず、ｂａｂｂｌｅＴｉｍｅｏｕｔ（２．２．１を参照）によって指定された時間枠内に認識なしで処理が停止した場合、音声認識プラットフォームによって投じられたイベントも処理する（ｉｉ）。

【０１３９】このイベントは、認識処理を自動的に取り消す。

構文：

【０１４０】

【表５】

インラインＨＴＭＬ	<reco onTimeout="handler" ...>
イベントプロパティ （ＥＣＭＡＳｃｒｉｐｔの場合）	Object.onTimeout = handler Object.onTimeout = GetRef("handler");

【０１４１】イベントオブジェクト情報：　【表６】

【０１４２】

41 42

バブル	なし
呼出し方法	認識が停止する前に maxtime 属性によって設定された時間枠が満了となった場合、ブラウザによって投じられる。
デフォルト動作	reco status を -12 に設定

【０１４３】イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。

【０１４４】２．４．４　ｏｎＮｏＲｅｃｏ：ｏｎＮｏＲｅｃｏは音声認識プラットフォームによって投じられたイベントが、有効な認識結果を戻すことができない場合に、ｓｔａｔｕｓコードによって区別される。イベントは認識プロセスを自動的に停止する。

＊合のハンドラである。発生する可能性のある様々なケース

【０１４５】構文：

【０１４６】

【表７】

インライン HTML	<Reco onNoReco="handler" >
イベントプロパティ	Object.onNoReco = handler; Object.onNoReco = GetRef("handler");

【０１４７】イベントオブジェクト情報：　※【表８】

【０１４８】

※

バブル	なし
呼出し方法	認識装置が音声を検出したが、発話を解釈することができない。
デフォルト動作	status プロパティを設定し、null 認識結果を戻す。status コードは以下のように設定される。 status -13：音声は検出されたが内容が解釈できない。 status -14：何らかの音声が検出され解釈されたが、確信値が不十分で拒否された（しきい値設定については２２１の reject 属性を参照）。 status -15：音声が検出され解釈されたが、音声の検出と babbleTimeout 属性に指定された待機期間との間に完全な認識を戻すことができなかった（２２１を参照）。

【０１４９】イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。

【０１５０】３　ｐｒｏｍｐｔ

ｐｒｏｍｐｔ要素は、システム出力を指定するのに使用される。そのコンテンツは、以下の１つまたは複数であってよい。

・インラインまたは参照済みテキスト。韻律的またはその他の音声出力情報でマークアップすることができる。

・含む側のドキュメントから提示時に取り出された変数値

・音声ファイルへのリンク

【０１５１】ｐｒｏｍｐｔ要素は、ダウンレベルブラウザによって、またはアップレベルブラウザのオブジェクトメソッドによって、宣言的に解釈される（またはＳＭＩＬコマンドによって活動化される）。

【０１５２】３．１　ｐｒｏｍｐｔのコンテンツ

ｐｒｏｍｐｔ要素は、テキストまたは音声ファイルの参照のいずれか、あるいはその両方としての、システム出力用のリソースを含む。単純なｐｒｏｍｐｔは、以下のように、出力用に必要なテキストを指定するだけでよい。

```
<prompt id="Welcome">
ＡＣＭＥ天気予報にお電話いただきありがとうございました。
</prompt>
```

この単純なテキストに、以下に記載する種類のいずれかのマークアップをさらに含めることができる。

【０１５３】３．１．１　音声合成マークアップ

任意のフォーマットの音声合成マークアップ言語を、ｐｒｏｍｐｔ要素内部で使用することができる。（このフォーマットは、３．２．１に記載した「ｔｔｓ」属性で指定することができる。）以下の例は、内部にある一定の語句を強調するための命令を備えたテキストを示すものである。

```
<prompt id="giveBalance">
口座に <emph> ５ドル </emph> 残金があります。
</prompt>
```


43 44

【０１５４】３．１．２　動的コンテンツ

ｐｒｏｍｐｔの実際のコンテンツは、ｐｒｏｍｐｔが出力される直前にクライアント側で計算する必要のある場合がある。特定の値を確認するために、たとえば変数内で値をデリファレンスする必要がある。その場合は、ｖａｌｕｅ要素を使用することができる。

【０１５５】Ｖａｌｕｅ要素

ｖａｌｕｅ：任意選択。ドキュメント内にある要素の値を取り出す。

【０１５６】属性：

・ｔａｒｇｅｔＥｌｅｍｅｎｔ：任意選択。ｈｒｅｆまたはｔａｒｇｅｔＥｌｅｍｅｎｔのいずれかを指定すること。取り出される値を含む要素のｉｄ。

・ｔａｒｇｅｔＡｔｔｒｉｂｕｔｅ：任意選択。値が取り出される要素の属性。

・ｈｒｅｆ：任意選択。音声セグメントのＵＲＩ。ｈｒｅｆとｔａｒｇｅｔＥｌｅｍｅｎｔの両方が存在する場合、ｈｒｅｆがｔａｒｇｅｔＥｌｅｍｅｎｔに優先される。

【０１５７】ｔａｒｇｅｔＥｌｅｍｅｎｔ属性は、含む側のドキュメント内にある要素の参照に使用される。ｉｄがｔａｒｇｅｔＥｌｅｍｅｎｔによって指定されている要素のコンテンツが、合成されるテキストに挿入される。所望のコンテンツが要素の属性内に保持されている＊場合、ｔａｒｇｅｔＡｔｔｒｉｂｕｔｅ属性を使用して、ｔａｒｇｅｔＥｌｅｍｅｎｔに必要な属性を指定することができる。これは、たとえばＨＴＭＬ書式制御の値をデリファレンスするのに役立つ。以下の例示では、ｐｒｏｍｐｔが出力される前に、「ｔｘｔＢｏｘＯｒｉｇｉｎ」要素および「ｔｘｔＢｏｘＤｅｓｔ」要素の「ｖａｌｕｅ」属性がテキストに挿入される。

```
<prompt id="Confirm">
    行く先は
    <value targetElement="txtBoxOrigin"
 targetAttribute="value" />
    から
    <value targetElement="txtBoxDest"
 targetAttribute="value" />
    ですか。
</prompt>
```

【０１５８】３．１．３　音声ファイル

ｖａｌｕｅ要素は、合成されたｐｒｏｍｐｔの代わりに、あるいはその中で再生するために、事前に録音された音声ファイルを参照するのに使用することもできる。以下の例では、ｐｒｏｍｐｔの終わりにビーッという音を鳴らす。

```
<prompt>
    ビープ音の後にメッセージを録音してください。
    <value bref="/wav/beep.wav" />
</prompt>
```

【０１５９】３．１．４　参照済みプロンプト

コンテンツをインラインで指定する代わりに、空の要素を備えたｓｒｃ属性を使用することで、以下のように、ＵＲＩを介して外部コンテンツを参照することができる。

```
<prompt id="Welcome" src="/ACMEWeatherPrompts#Welcome"/>
```

ｓｒｃ属性のターゲットは、インラインプロンプト用に指定された上記コンテンツのいずれかまたはすべてを保持することができる。

【０１６０】３．２　属性およびプロパティ

ｐｒｏｍｐｔ要素は、以下の属性（ダウンレベルブラウザ）およびプロパティ（ダウンレベルおよびアップレベルブラウザ）を保持する。

【０１６１】３．２．１　属性

・ｔｔｓ：任意選択。テキストから音声への合成に関するマークアップ言語タイプ。デフォルトの値は「ＳＡＰＩ　５」。

・ｓｒｃ：インラインプロンプトが指定された場合は任意選択。参照済みプロンプトのＵＲＩ（３．１．４を参照）。

・ｂａｒｇｅｉｎ：任意選択。整数。プロンプトの開始から聴取者が再生に割り込んだときまでの、ミリ秒単位の時間枠。デフォルトの値は無限であり、すなわちｂａｒｇｅｉｎは使用できない。Ｂａｒｇｅｉｎ＝０の場合は、即時ｂａｒｇｅｉｎが可能である。これは、プラットフォームがサポートしているどんな種類の割り込みにも適用される。ｒｅｃｏが開始された時点で、キーワードまたはエネルギーベースのｂａｒｇｅｉｎ時間のどちらが使用可能であるかに応じて、いずれかがこの方法で構成される。

・ｐｒｅｆｅｔｃｈ：任意選択。ページがロードされたときに、ｐｒｏｍｐｔをブラウザ側で即時に合成およびキャッシュするべきであるかどうかを示す、ブールフラグ。デフォルトの値はｆａｌｓｅである。

【０１６２】３．２．２　プロパティ

アップレベルブラウザは、ｐｒｏｍｐｔのＤＯＭオブジェクト内で以下のプロパティをサポートする。

・ｂｏｏｋｍａｒｋ：読取り専用。発生した最新の合成ブックマークのテキストを録音する文字列オブジェクト。

・ｓｔａｔｕｓ：読取り専用。音声プラットフォームによって戻されるｓｔａｔｕｓコード。

・ｉｎｎｅｒｔｅｘｔ：読取り専用。このプロパティ

45

は、ｐｒｏｍｐｔのテキスト表現形式（ｔｒａｎｓｃｒｉｐｔｉｏｎ）を提供し、これが合成装置に送られる。たとえば、ｐｒｏｍｐｔが可聴波ファイルの再生を含む場合、このプロパティはそのｐｒｏｍｐｔのテキストバージョン（可聴波ファイルと共に格納されている場合が多い）を提供し、次いでたとえば、ｐｒｏｍｐｔのテキストバージョンを構成要素またはクライアントデバイス上で実行中のアプリケーションに提供することによって、表示されるか、そうでなければ使用される。ｉｎｎｅｒｔｅｘｔプロパティは、動的コンテンツを含むｐｒｏｍｐｔのテキストバージョンを提供する際にも使用できる。

【０１６３】３．３　ｐｒｏｍｐｔメソッド

ｐｒｏｍｐｔ再生は、ｐｒｏｍｐｔのＤＯＭオブジェクトにある以下のメソッドを使用して制御することができる。この方法では、アップレベルブラウザが、ｐｒｏｍｐｔオブジェクトの開始（ｓｔａｒｔ）および停止（ｓｔｏｐ）、進行中のｐｒｏｍｐｔの一時停止（ｐａｕｓｅ）および再開（ｒｅｓｕｍｅ）、ならびに合成された音声の速度および音量の変更（ｃｈａｎｇｅ）を実行することができる。

【０１６４】３．３．１　Ｓｔａｒｔ

ｐｒｏｍｐｔの再生を開始する。引数が与えられない限り、メソッドはオブジェクトのコンテンツを再生する。単一のｐｒｏｍｐｔオブジェクトのみが所与の時間に「開始された」とみなされるため、Ｓｔａｒｔが連続して呼び出されると、すべての再生が連続して行われる。

【０１６５】構文：

Ｏｂｊｅｃｔ．Ｓｔａｒｔ（[ｓｔｒＴｅｘｔ]）；

パラメータ：

・ｓｔｒＴｅｘｔ：合成装置に送信されるテキスト。存在する場合は、この引数がオブジェクトのコンテンツに優先される。

戻り値：

なし

例外：ｓｔａｔｕｓ＝－１に設定すると、音声バッファがサーバによってすでに解除されている場合はｏｎＣｏｍｐｌｅｔｅイベントが開始される。

【０１６６】３．３．２　Ｐａｕｓｅ

音声バッファをフラッシングせずに再生を一時停止する。再生が一時停止または停止されている場合、このメソッドは無効である。

【０１６７】構文：

Ｏｂｊｅｃｔ．Ｐａｕｓｅ（）；

戻り値：

なし

46

例外：

なし

【０１６８】３．３．３　Ｒｅｓｕｍｅ

音声バッファをフラッシングせずに再生を再開する。再生が一時停止されていない場合、このメソッドは無効である。

【０１６９】構文：

Ｏｂｊｅｃｔ．Ｒｅｓｕｍｅ（）；

戻り値：

なし

例外：再開が失敗すると例外が投じられる。

【０１７０】３．３．４　Ｓｔｏｐ

すでに停止されていなければ再生を停止し、音声バッファをフラッシングする。再生がすでに停止されている場合、このメソッドは音声バッファをフラッシングするだけである。

【０１７１】構文：

Ｏｂｊｅｃｔ．Ｓｔｏｐ（）；

戻り値：

なし

例外：

なし

【０１７２】３．３．５　Ｃｈａｎｇｅ

再生の速度および／または音量を変更する。Ｃｈａｎｇｅは再生中に呼び出すことができる。

【０１７３】構文：

Ｏｂｊｅｃｔ．Ｃｈａｎｇｅ（ｓｐｅｅｄ，ｖｏｌｕｍｅ）；

パラメータ：

・ｓｐｅｅｄ：必須。変更する因数。ｓｐｅｅｄ＝２．０は現在の速度を２倍にする、ｓｐｅｅｄ＝０．５は現在の速度を半分にする、ｓｐｅｅｄ＝０はデフォルト値を復元するという意味である。

・ｖｏｌｕｍｅ：必須。変更する因数。ｖｏｌｕｍｅ＝２．０は現在の音量を２倍にする、ｖｏｌｕｍｅ＝０．５は現在の音量を半分にする、ｖｏｌｕｍｅ＝０はデフォルト値を復元するという意味である。

戻り値：

なし

例外：

なし

【０１７４】３．３．６　ｐｒｏｍｐｔの制御例

以下の例は、上記のメソッドを使用して、キーワード割込み機構をサポートしていないプラットフォームに対して、ｐｒｏｍｐｔの制御をどのようにオーサリングできるかを示すものである。

47

```
<html>
<title> プロンプト制御 </title>
<head>
  <script>
  <!--
      function checkKWBargein() {
        news.change(1.0, 0.5); // 確認中ボリュームを下げる。
        if (keyword.text == "") { // 結果は閾値未満である。
            news.change(1.0, 2.0); // ボリュームを元に戻す。
            keyword.Start(); // 認識を再開する。
        } else {
            news.Stop(); // キーワードが検出された。プロンプトを停止する。
            // 必要なことを行う。
        }
      }
  //
  </script>
  <script for="window" event="onload">
  <!--
      news.Start(); keyword.Start();
  //
  </script>
  </head>
  <body>
  <prompt id="news" bargein="0">
```

【０１７５】水曜日、来週の連邦準備会議を前に、投資家が何らかの大きな動きに出るにはほとんど刺激を受けなかったため、株は新たなパフォーマンスの低迷に入った。ハイテク株比率の高いナスダック総合指数は４２．＊

＊５１ポイント下がって２１５６．２６で引けた。ダウ平均株価は、午後の早い時間に反発できなかった後、１７．０５ポイント下落して１０８６６．４６になった。

```
  - <!--
  </prompt>
  <reco     id="keyword"
      reject="70"
      onReco="checkKWBargein()" >
      <grammar src=http://denali/news_bargein_grammar.xml />
  </reco>
  </body>
  </html>
```

【０１７６】３．４　Ｐｒｏｍｐｔイベント

ｐｒｏｍｐｔ　ＤＯＭオブジェクトは以下のイベントをサポートしており、そのハンドラは、ｐｒｏｍｐｔ要素の属性として指定することができる。

【０１７７】３．４．１　ｏｎＢｏｏｋｍａｒｋ　※40

※合成ブックマークに遭遇すると開始される。このイベントは再生を一時停止しない。

【０１７８】構文：

【０１７９】

【表９】

インライン HTML	<prompt onBookmark="handler"...>
イベントプロパティ	Object.onBookmark = handler Object.onBookmark = GetRef("handler");

【０１８０】イベントオブジェクト情報：

【０１８１】

【表１０】

バブル	なし
呼出し方法	提示された文字列内のブックマークに遭遇する
デフォルト動作	ブックマーク文字列を戻す

49

【０１８２】イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。

【０１８３】３．４．２　ｏｎＢａｒｇｅｉｎ：ユーザのｂａｒｇｅ－ｉｎイベントが検出されると開始される。（ｂａｒｇｅ－ｉｎイベントの構成内容、たとえ

50

＊ば、エネルギー検出またはキーワード認識を決定することはプラットフォームの責任であることに留意されたい。）このイベントハンドラを指定しても、ｂａｒｇｅ－ｉｎは自動的にオンにならない。構文：

【０１８４】

【表１１】

インラインHTML	<prompt onBargein="handler"...>
イベントプロパティ	Object.onBargein = handler Object.onBargein = GetRef("handler");

【０１８５】イベントオブジェクト情報：　※【表１２】

【０１８６】

※

バブル	なし
呼出し方法	bargeinイベントに遭遇する
デフォルト動作	なし

【０１８７】イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。

【０１８８】３．４．３　ｏｎＣｏｍｐｌｅｔｅ：ｐｒｏｍｐｔ再生が終わりに達するか、または例外（上記に★

20

★定義）に遭遇すると、開始される。

【０１８９】構文：

【０１９０】

【表１３】

インラインHTML	<prompt onComplete="handler"...>
イベントプロパティ	Object. onComplete = handler Object. onComplete = GetRef("handler");

【０１９１】イベントオブジェクト情報：　☆【表１４】

【０１９２】

☆

バブル	なし
呼出し方法	prompt 再生が完了する
デフォルト動作	再生が通常通りに完了した場合は status = 0に設定するが、そうでなければ status を上記で指定されたように設定する。

【０１９３】イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。

【０１９４】３．４．４　ブックマークおよびイベントの使用

以下の例は、ｐｒｏｍｐｔ出力中にｂａｒｇｅｉｎが発生した場合に関して、ユーザの応答の意味、すなわち出発都市の修正または行先都市の準備のいずれであるかを特定するために、ブックマークイベントをどのように使用するかを示す例である。ｏｎＢａｒｇｅｉｎハンドラが、グローバル「ｍａｒｋ」変数をｐｒｏｍｐｔ内で遭遇した最新のブックマークに設定するスクリプトを呼び出し、この「ｍａｒｋ」の値が正しい値を設定するためにｒｅｃｏの後処理関数（「ｈｅａｒｄ」）で使用される。

```
<script><![CDATA[
    var mark;
    function interrupt( ) {
        mark = event.srcElement.bookmark;
    }
    function ProcessCityConfirm() {
        confirm.stop(); // オーディオバッファをフラッシュする。
        if (mark == "mark_origin_city")
            txtBoxOrigin.value =
event.srcElement.text;
        else
```


51

```
            txtBoxDest.value =
event.srcElement.text;
    ]
]]></script>
<body>
<input name="txtBoxOrigin" value="シアトル"
type="text"/>
<input name="txtBoxDest" type="text" />
...
<prompt id="confirm" onBargein="interrupt()"
bargein="0">
    From <bookmark mark="mark_origin_city" />
    <value targetElement="orgin"
targetAttribute="value" />.
    <bookmark mark="mark_dest_city" /> が行く先であることを復唱してください。
</prompt>
<reco onReco="ProcessCityConfirm()" >
    <grammar src="/grm/1033/cities.xml" />
</reco>
...
</body>
```

52

【０１９５】４　ＤＴＭＦ

ＤＴＭＦ認識オブジェクトを作成する。このオブジェクトはインラインマークアップ言語構文を使用するかスクリプティングでインスタンス化することができる。ＤＴＭＦは活動化されたときに、ｐｒｏｍｐｔオブジェクトにｂａｒｇｅ－ｉｎイベントを開始させることができる。ＤＴＭＦ認識に関して以下で論じるタグおよびイベンティング、ならびに第５項で論じる呼制御は、一般に、音声ブラウザ２１６とメディアサーバ２１４との間の対話に関係するものである。

【０１９６】４．１　コンテンツ

・ｄｔｍｆｇｒａｍｍａｒ：インライン文法用である。

例１：キーをテキストにマッピングする

```
<input type="text" name="city"/>
<DTMF id="city_choice" timeout="2000" numDigits="1">
    <dtmfgrammar>
        <key value="1"> シアトル </key>
        <key value="2"> ボストン </key>
    </dtmfgrammar>
    <bind targetElement="city" targetAttribute="value"
/>
</DTMF>
```

「ｃｉｔｙ＿ｃｈｏｉｃｅ」が活動化されたときに、ユーザが１を押すと「Ｓｅａｔｔｌｅ」が、２を押すと「Ｂｏｓｔｏｎ」が入力フィールドに割り当てられ、それ以外の場合は何も割り当てられない。

【０１９８】

・ｂｉｎｄ：ＤＴＭＦ変換結果を適切なフィールドに割り当てる。

【０１９７】属性：

・ｔａｒｇｅｔＥｌｅｍｅｎｔ：必須。部分的な認識結果が割り当てられる要素（たとえばＷ３Ｃ　ＳＭＩＬ２．０の場合と同様）。

・ｔａｒｇｅｔＡｔｔｒｉｂｕｔｅ：認識結果が割り当てられるターゲット要素の属性（たとえばＳＭＩＬ２．０の場合と同様）。デフォルトの値は「ｖａｌｕｅ」。

・ｔｅｓｔ：割当ての条件。デフォルトの値はｔｒｕｅ。

53

例２：複数フィールドの場合のＤＴＭＦの使用方法

```
〈input type="text" name="area_code" /〉
〈input type="text" name="phone_number" /〉
〈DTMF id="areacode" numDigits="3"
onReco="extension.Activate()"〉
    〈bind targetElement="area_code" /〉
〈/DTMF〉
〈DTMF id="extension" numDigits="7"〉

    〈bind targetElement="phone_number" /〉
〈/DTMF〉
```

この例は、ユーザが複数のフィールドに入力する方法を＊

例３：音声入力とＤＴＭＦ入力の両方を使用可能にし、ユーザがＤＴＭＦを開始した場合は音声を使用不能にする方法

```
<input type="text" name="credit_card_number" />
<prompt onBookmark="dtmf.Start(); speech.Start()"
      bargein="0">
    <bookmark name="starting" />
    を復唱するか、クレジットカード番号を入力してください。
</prompt>
<DTMF id="dtmf" escape="#" length="16"
interdigitTimeout="2000"
    onkeypress="speech.Stop()">
    <bind targetElement="credit_card_number" />
</DTMF>
<reco id="speech" >
    <grammar src="/grm/1033/digits.xml" />
    <bind targetElement="credit_card_number" />
</reco>
```

【０２００】４．２　属性およびプロパティ

４．２．１　属性

・ｄｔｍｆｇｒａｍｍａｒ：必須。ＤＴＭＦ文法のＵＲＩ。

【０２０１】４．２．２　プロパティ

・ＤＴＭＦｇｒａｍｍａｒ：読取り／書込み

ＤＴＭＦを表すＸＭＬ　ＤＯＭ　Ｎｏｄｅオブジェクトから文字列への変換マトリックス（ＤＴＭＦ文法とも呼ばれる）。デフォルトの文法は以下のとおり。

```
〈dtmfgrammar〉
    〈key value="0"〉0〈/key〉
    〈key value="1"〉1〈/key〉
    ...
    〈key value="9"〉9〈/key〉
    〈key value="*"〉*〈/key〉
    〈key value="#"〉#〈/key〉
〈/dtmfgrammar〉
```

・ｆｌｕｓｈ

活動化の前に基礎となる電話インターフェースカード上でＤＴＭＦバッファを自動的にフラッシングさせるかどうかを示す、読取り／書込みのブールフラグ。デフォルトの値は、先行タイプを実行可能にするためのｆａｌｓｅ。

・ｅｓｃａｐｅ

読取り／書込み。ＤＴＭＦ読取りセッションを終了するためのｅｓｃａｐｅキー。ｅｓｃａｐｅキーは１つのキーである。

・ｎｕｍＤｉｇｉｔｓ

読取り／書込み。ＤＴＭＦ読取りセッションを終了するためのキーストロークの数。ｅｓｃａｐｅとｌｅｎｇｔｈの両方が指定された場合、どちらかの条件が満たされるとＤＴＭＦセッションは終了する。

・ｄｔｍｆＲｅｓｕｌｔ

ユーザが入力したＤＴＭＦを格納している読取り専用文字列。入力された場合は結果的にｅｓｃａｐｅが含まれる。

・ｔｅｘｔ

空白で分離されたトークン文字列を格納している読取り専用文字列であって、各トークンはＤＴＭＦ文法に従って変換される。

・ｉｎｉｔｉａｌＴｉｍｅｏｕｔ

読取り／書込み。第１のＤＴＭＦキーストロークを受け

55

取るための、ミリ秒単位のタイムアウト時間枠。指定されない場合、デフォルトの値は電話プラットフォームの内部設定となる。

・ｉｎｔｅｒｄｉｇｉｔＴｉｍｅｏｕｔ

読取り／書込み。隣接するＤＴＭＦキーストロークのための、ミリ秒単位のタイムアウト時間枠。指定されない場合、デフォルトの値は電話プラットフォームの内部設定となる。

【０２０２】４．３　オブジェクトメソッド：

４．３．１　Ｓｔａｒｔ

ＤＴＭＦ割込みを実行可能にし、ＤＴＭＦ読取りセッションを開始する。

【０２０３】構文：

Ｏｂｊｅｃｔ．Ｓｔａｒｔ（）；

戻り値：

なし

例外：

なし

【０２０４】４．３．２　Ｓｔｏｐ

ＤＴＭＦを使用不能にする。ただし、ユーザが入力したキーストロークは、バッファ内に残る。

【０２０５】構文：

Ｏｂｊｅｃｔ．Ｓｔｏｐ（）；

【０２０６】４．３．３　Ｆｌｕｓｈ

ＤＴＭＦバッファをフラッシングする。ＦｌｕｓｈはＤＴＭＦセッション中に呼び出すことができない。

【０２０７】構文：

Ｏｂｊｅｃｔ．Ｆｌｕｓｈ（）；

戻り値：

なし

例外：

なし

【０２０８】４．４　イベント

４．４．１　ｏｎｋｅｙｐｒｅｓｓ

ＤＴＭＦキーを押すと開始される。これはＨＴＭＬ制御から引き継がれたデフォルトイベントに優先する。ユーザがｅｓｃａｐｅキーを叩くと、ｏｎＫｅｙｐｒｅｓｓではなくｏｎＲｅｃイベントが開始される。

【０２０９】構文：

【０２１０】

【表１５】

＊

インラインHTML	<DTMF onkeypress="handler" ...>
イベントプロパティ	Object.onkeypress = handler Object.onkeypress = GetRef("handler");

【０２１１】イベントオブジェクト情報：　※【表１６】

【０２１２】

※

バブル	なし
呼出し方法	タッチトーン電話キーパッドを押す
デフォルト動作	押されたキーを戻す

【０２１３】イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。

【０２１４】４．４．２　ｏｎＲｅｃｏ

ＤＴＭＦセッションが終了すると開始される。このイベ★

★ントは現在のＤＴＭＦオブジェクトを自動的に使用不能にする。

【０２１５】構文：

【０２１６】

【表１７】

インラインHTML	<DTMF onReco="handler" ...>
イベントプロパティ	Object.onReco = handler Object.onReco = GetRef("handler");

【０２１７】イベントオブジェクト情報：　☆【表１８】

【０２１８】

☆

バブル	なし
呼出し方法	ユーザがescapeキーを押すか、またはキーストローク数が指定された値に合致する
デフォルト動作	押されたキーを戻す

【０２１９】イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。

【０２２０】４．４．３　ｏｎＴｉｍｅｏｕｔ

タイムアウト以前に句完了イベントが受け取られないと開始される。このイベントは認識プロセスを自動的に停

止する。

【０２２１】構文：

＊【０２２２】

＊【表１９】

インラインHTML	<DTMF onTimeout="handler"...>
イベントプロパティ (ECMAScriptの場合)	Object.onTimeout = handler Object.onTimeout = GetRef("handler");

【０２２３】イベントオブジェクト情報：

※【表２０】

【０２２４】

※

バブル	なし
呼出し方法	指定されたタイムアウト内に、何のDTMFキーストロークも検出されない
デフォルト動作	なし

【０２２５】イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。

【０２２６】５ Ｃａｌｌｃｏｎｔｒｏｌオブジェクト

電話音声ブラウザの電話インターフェース（呼、端末、および接続）を表す。このオブジェクトは、ＧＵＩブラウザでのｗｉｎｄｏｗオブジェクトと同じように固有である。したがって、電話オブジェクトの存続期間はブラウザインスタンスそれ自体と同じである。電話用の音声ブラウザは、呼ごとに１つずつ電話オブジェクトをインスタンス化する。ユーザはオブジェクトのインスタンス化または配置は行わない。

【０２２７】ここでは、このオブジェクトを介して第１の当事者の呼制御に関する特徴だけを示す。

【０２２８】５．１ プロパティ

・ａｄｄｒｅｓｓ

読取り専用。ＸＭＬ ＤＯＭノードオブジェクト。実施形態によって異なる。発呼者のアドレスである。ＰＳＴＮの場合は、ＡＮＩとＡＬＩの組合せであってよい。ＶｏＩＰの場合、これが発呼者のＩＰアドレスである。・ｒｉｎｇｓＢｅｆｏｒｅＡｎｓｗｅｒ着信呼に応答する前の呼出し音の回数。デフォルト値は無限であり、これは、開発者が呼に応答するために、具体的には以下のＡｎｓｗｅｒ（）メソッドを使用しなければならないことを意味する。電話局が着信呼を待ち行列に入れるのにＡＣＤを使用している場合、この番号は０に設定することができる。

【０２２９】５．２ メソッド

注：ここに示すメソッドはすべて同時に発生するものである。

【０２３０】５．２．１ Ｔｒａｎｓｆｅｒ

呼を転送する。ブラインド転送の場合、転送が完了すると、システムはオリジナルの呼およびフリーシステムリソースを終了することができる。

構文：

ｔｅｌｅｐｈｏｎｅ．Ｔｒａｎｓｆｅｒ（ｓｔｒＴｅｘｔ）；

パラメータ：

・ｓｔｒＴｅｘｔ：必須。所期の受信者のアドレス。

戻り値：

なし

例外：呼転送が失敗すると例外が投じられる。たとえば、最終当事者が話中の場合、こうした番号のファックスまたは留守番電話は応答しない。

【０２３１】５．２．２ Ｂｒｉｄｇｅ

第三者転送。呼が転送された後、ブラウザは呼に割り振られたリソースを解除することができる。転送された呼がｓｔｒＵＩＤを使用して戻されるときに、セッション状態を回復させるのはアプリケーションの役割である。基礎となる電話プラットフォームは、戻ってくる呼の宛先を異なるブラウザに指定することができる。呼は、受取り側が呼を終了させたときにのみ戻すことができる。

【0,２３２】構文：

ｔｅｌｅｐｈｏｎｅ．Ｂｒｉｄｇｅ（ｓｔｒＴｅｘｔ，ｓｔｒＵＩＤ，［ｉｍａｘＴｉｍｅ］）；

パラメータ：

・ｓｔｒＴｅｘｔ：必須。所期の受信者のアドレス。

・ｓｔｒＵＩＤ：必須。現在の呼を一意に識別するセッションＩＤ。転送された呼が戻されるときに、ｓｒｔＵＩＤがａｄｄｒｅｓｓ属性に表示される。

・ｉｍａｘＴｉｍｅ：任意選択。転送された呼の秒単位の最大持続時間。指定されない場合、デフォルトの値はプラットフォーム内部値である。

戻り値：

なし

例外：

なし

【０２３３】５．２．３ Ａｎｓｗｅｒ

呼に応答する。

構文：

ｔｅｌｅｐｈｏｎｅ．Ａｎｓｗｅｒ（）

戻り値：

なし

例外：接続がない場合に例外が投じられる。この場合、ｏｎＡｎｓｗｅｒイベントは開始されない。

【０２３４】５．２．４ Ｈａｎｇｕｐ

呼を終了させる。現在進行中の呼がない場合は無効である。

構文：

ｔｅｌｅｐｈｏｎｅ．Ｈａｎｇｕｐ（）

戻り値：

なし

例外：

なし

【０２３５】５．２．５ Ｃｏｎｎｅｃｔ

第１当事者の外に向かう呼を開始する。

構文：

ｔｅｌｅｐｈｏｎｅ．Ｃｏｎｎｅｃｔ（ｓｔｒＴｅｘｔ，［ｉＴｉｍｅｏｕｔ］）；

パラメータ：

・ｓｔｒＴｅｘｔ：必須。所期の受信者のアドレス。

・ｉＴｉｍｅｏｕｔ：任意選択。試行を中止するまでのミリ秒単位の時間。

指定されない場合、デフォルトの値はプラットフォーム内部値である。

戻り値：

なし

例外：話中信号に遭遇するか、あるいはＦＡＸまたは留守番電話に到達した場合を含み、呼が完了できない場合に例外が投じられる（注：ハードウェアはこの機能をサポートできない）。

【０２３６】５．２．６ Ｒｅｃｏｒｄ

ユーザの音声をファイルに記録する。

構文：

ｔｅｌｅｐｈｏｎｅ．Ｒｅｃｏｒｄ（ｕｒｌ，ｅｎｄＳｉｌｅｎｃｅ，［ｍａｘＴｉｍｅｏｕｔ］，［ｉｎｉｔｉａｌＴｉｍｅｏｕｔ］）；

```
<HTML>
<HEAD>
  <TITLE>Logon Page</TITLE>
 </HEAD>
  <SCRIPT>
   var focus;
   function RunSpeech() {
       if (logon.user.value == "") {
            focus="user";
            p_uid.Start(); g_login.Start(); dtmf.Start();
   return;
       }
       if (logon.pass.value == "") {
            focus="pin";
              p_pin.Start(); g_login.Start(); dtmf.Start();
    return;
       }
       p_thank.Start(); logon.submit();
   }
```

パラメータ：

・ｕｒｌ：必須。記録された結果のｕｒｌ

・ｅｎｄＳｉｌｅｎｃｅ：必須。無音が検出された後に記録を停止するまでのミリ秒単位の時間。

・ｍａｘＴｉｍｅｏｕｔ：任意選択。記録するための秒単位の最大時間。デフォルトの値はプラットフォームによって異なる。

・ｉｎｉｔｉａｌＴｉｍｅｏｕｔ：任意選択。記録の始めの部分で可能な無音の最大時間（ミリ秒単位）。

戻り値：

なし

例外：記録がｕｒｌに書き込めないときに、例外が投じられる。

【０２３７】５．３ イベントハンドラ

電話音声ブラウザを使用するアプリケーション開発者は、以下のイベントハンドラを実施することができる。

【０２３８】５．３．１ ｏｎＩｎｃｏｍｉｎｇ（）

音声ブラウザが着信呼を受け取ったときに呼び出される。すべての開発者はこのハンドラを使用して、呼に応答する前に発呼者アドレスを読み取り、カスタマイズされた機能を呼び出すことができる。

【０２３９】５．３．２ ｏｎＡｎｓｗｅｒ（）

音声ブラウザが着信呼に応答したときに呼び出される。

【０２４０】５．３．３ ｏｎＨａｎｇｕｐ（）

ユーザが電話を切ったときに呼び出される。このイベントは、プログラムがＨａｎｇｕｐまたはＴｒａｎｓｆｅｒメソッドを呼び出したときに自動的に開始されるものではない。

【０２４１】５．４ 例

以下の例は、電話セッションを処理するために呼制御イベントに送られるスクリプティングを示すものである。

61

```
function login_reco() {
    res = event.srcElement.recoResult;
    pNode = res.selectSingleNode("//uid");
    if (pNode != null)
        logon.user.value = pNode.xml;
    pNode = res.selectSingleNode("//password");
    if (pNode != null)
        logon.pass.value = pNode.xml;
}
function dtmf_reco() {
    res = event.srcElement.dtmfResult;
    if (focus == "user")
        logon.user.value = res;
    else
        logon.pin.value = res;
}
</SCRIPT>
<SCRIPT for="callControl" event="onIncoming">
  <!--
    // 読取りアドレス。カスタマイズされたものがあれば準備する。
    callControl.Answer();
  //
</SCRIPT>
<SCRIPT for="callControl" event="onOffhook">
  <!--
    p_main.Start(); g_login.Start(); dtmf.Start();
  focus="user";
  //
</SCRIPT>
<SCRIPT for="window" event="onload">
  <!--
      if (logon.user.value != "") {
          p_retry.Start();
          logon.user.value = "";
          logon.pass.value = "";
          checkFields();
      }
  //
</SCRIPT>
<BODY>
<reco id="g_login"
    onReco="login_reco(); runSpeech()"
    timeout="5000"
    onTimeout="p_miss.Start(); RunSpeech() >
    <grammar
 src=http://kokaneel/etradedemo/speechonly/login.xml/>
    </ reco >
<dtmf id="dtmf"
    escape="#"
    onkeypress="g_login.Stop();"
```


62

63

```
    onReco="dtmf_reco();RunSpeech()"
    interdigitTimeout="5000"
    onTimeout="dtmf.Flush(); p_miss.Start();RunSpeech()" />

<prompt id="p_main"> ユーザＩＤおよびピン番号を復唱してください。</pro
mpt>
<prompt id="p_uid"> ユーザＩＤだけ復唱してください。</prompt>
<prompt id="p_pin"> ピン番号だけ復唱してください。</prompt>
<prompt id="p_miss"> 申し訳ありません。失敗しました。</prompt>
<prompt id="p_thank"> ありがとうございました。身元を確認する間お待ちく
ださい。</prompt>
<prompt id="p_retry"> 申し訳ありません。ユーザＩＤおよびピン番号が一致
しません。</prompt>
<H2>Login</H2>
<form id="logon">
    UID: <input name="user" type="text"
    onChange="runSpeech()" />
    PIN: <input name="pass" type="password"
    onChange="RunSpeech()" />
</form>
</BODY>
</HTML>
```


64

【０２４２】６　ダイアログフローの制御

６．１　ダイアログフローを実施するためのＨＴＭＬおよびスクリプトの使用この例は、入力ボックスに入れる値を探し、入力用の文脈依存ヘルプを提供する、単純なダイアログフローを実施する方法を示すものである。ヘルププロンプトのコンテンツ形成を助けるためにＨＴＭＬ入力機構（「ｔｏｏｌｔｉｐ」機構として装示ブラウザで使用される）で、ｔｉｔｌｅ属性を使用する。

```
<html>
  <title>Context Sensitive Help</title>
<head>
  <script>   var focus;
      function RunSpeech() {
          if (trade.stock.value == "") {
              focus="trade.stock";
              p_stock.Start();
              return;
          }
          if (trade.op.value == "") {
              focus="trade.op";
              p_op.Start();
              return;
          }
          //.. フィールドすべてについて上記を繰り返す。
          trade.submit();
      }
      function handle() {
          res = event.srcElement.recoResult;
          if (res.text == "help") {
              text = "復唱してください";
              text += document.all[focus].title;
              p_help.Start(text);
```

```
        } else {
            // 値割当てを続行する。
            }
        }
    </script>
    </head>
<body>
    <prompt id="p_help" onComplete="checkFileds()" />
    <prompt id="p_stock" onComplete="g_stock.Start()"> 株式名を復唱してください。</prompt>
    <prompt id="p_op" onComplete="g_op.Start()"> 売りですか。買いですか。</prompt>
    <prompt id="p_quantity" onComplete="g_quantity.Start()"> 何株ですか。</prompt>
    <prompt id="p_price" onComplete="g_price.Start()"> 価格はいくらですか。</prompt>
    <reco id="g_stock" onReco="handle(); checkFields()" >
        <grammar src="./g_stock.xml" />
    </ reco >
    <reco id="g_op" onReco="handle(); checkFields()" />
        <grammar src="./g_op.xml" />
    </ reco >
    <reco id="g_quantity" onReco="handle(); checkFields()" />
        <grammar src="./g_quant.xml" />
    </ reco >
    <reco id="g_price" onReco="handle(); checkFields()" />
        <grammar src="./g_quant.xml" />
    </reco>
    <form id="trade">
        <input name="stock" title="stock name" />
        <select name="op" title="buy or sell">
            <option value="buy" />
            <option value="sell" />
            </select>
        <input name="quantity" title="number of shares" />
        <input name="price" title="price" />
    </form>
    </body>
    </html>
```

６．２　ＳＭＩＬの使用

以下の例は、ＳＭＩＬ機構を使用したｐｒｏｍｐｔおよびｒｅｃｏ要素の活動化の例を示すものである。

```
    <html xmlns:t="urn:schemas-microsoft-com:time"
          xmlns:sp="urn:schemas-microsoft-com:speech">
    <head>
    <style>
      .time { behavior: url(#default#time2); }
    </style>
    </head>
    <body>
```



```
    <input name="txtBoxOrigin" type="text"/>
    <input name="txtBoxDest" type="text" />
    <sp:prompt class="time" t:begin="0">
        出発地と行き先の都市を復唱してください。
    </sp:prompt>
    <t:par t:begin="time.end" t:repeatCount="indefinitely"
      <sp:reco class="time" >
        <grammar src="./city.xml" />
        <bind targetElement="txtBoxOrigin"
         value="//origin_city" />
        <bind targetElement="txtBoxDest"
         test="/sml/dest_city[@confidence $gt$ 40]"
         value="//dest_city" />
      </sp:reco>
    </t:par>
    </body>
    </html>
```

【０２４３】７　ＳＭＥＸ（Ｍｅｓｓａｇｅ）要素／オブジェクト

Ｓｉｍｐｌｅ　Ｍｅｓｓａｇｉｎｇ　ＥＸｃｈａｎｇｅ／ＥＸｔｅｎｓｉｏｎの短縮形であるＳＭＥＸは、クライアントデバイスのプラットフォーム上にある外部の構成要素またはアプリケーションと通信するオブジェクトである。タグ名＜ｓｍｅｘ＞を備えた要素として、ＸＭＬまたは同様のマークアップベースドキュメントに埋め込むことができる。メッセージングオブジェクトの例示的な使用法には、ロギングおよび電話制御を含まれることがある。オブジェクトは、マークアップベースの認識、およびメッセージングを介して新しい機能を追加できるようにするプロンプティングの拡張可能性を裏すものである。

【０２４４】インスタンス化すると、オブジェクトは、その構成パラメータまたは属性の指定を介して、プラットフォーム構成要素またはアプリケーションを備えた非同期メッセージ交換チャネルを確立するように指示される。オブジェクトは、プロパティが割当て動作の受取り側である場合に必ず、コンテンツがプラットフォーム構成要素またはアプリケーションに送信される文字列プロパティを有する（すなわち、ｖａｌｕｅ）。同様に、プラットフォーム構成要素またはアプリケーションから受け取ったメッセージを保持する、ＸＭＬ　ＤＯＭ　Ｎｏｄｅタイプのプロパティも有する。ｍｅｓｓａｇｅオブジェクトは、プラットフォームメッセージを受け取ると必ずイベントを送信する。オブジェクトは、その基本動作が非同期であるため、タイムアウト設定を処理するためのアプリケーション開発者用内蔵クロックも有する。

【０２４５】ｍｅｓｓａｇｅまたはｓｍｅｘオブジェクトは、通信手段に対して寛容である。ただし、一実施形態では、ｓｍｅｘオブジェクトは通常のＸＭＬまたはマークアップ要素と同じ持続期間を有するものであり、すなわち、ｓｍｅｘオブジェクトはそのホストドキュメントがアンロードされると廃棄される。多くの場合、ｓｍｅｘオブジェクトは自動クリーンアップを実行し、アンロードされたときに通信リソースを解放することができるため、マークアップページ間に渡って持続通信リンクが望ましい場合（たとえば呼制御）に、使用されることがある。こうした場合に、アーキテクチャは、割り振られたリソースを放棄する（たとえば、ソケットを閉じる）責務をアプリケーション開発者に負わせる。

【０２４６】ｓｍｅｘオブジェクトは、メッセージの形態（スキーマ）に関して中立である。一部の実施形態では、（ＳＩＰまたはＣＣＸＭＬで使用されているような）既存の標準メッセージフォーマットを強く優先させ、いくつかの基本的なスキーマをサポートするように実施者に対して要求することが望ましい場合がある。本来アーキテクチャは、プラットフォームとアプリケーションの両方の開発者が、その間に相互運用性を失うことなく他の機能を導入するために、ＸＭＬまたは同様のマークアップの標準化された拡張可能性をフルに利用できるようにするものである。

【０２４７】

例１：ｓｍｅｘをロギングオブジェクトとして使用する

```
<smex id="logServer">
    <param name="d:server" xmlns:d="urn:Microsoft.com/COM">
        <d:protocol>DCOM</d:protocol>
        <d:clsid>209309302930202932094209843209 8</d:clsid>
```

```
        <d:iid>09038593049034985309853090948O3</d:iid>
    </param>
</smex>
<listen ...>
    ...// 入力フィールドのｒｅｃｏ結果を拘束する他の指示
    <bind targetElement="logServer" targetAttribute="sent"
        value="*[@log $ge$ 3]"/>
</listen>
```

【０２４８】この例は、クラスｉｄおよびインターフェースｉｄを備えたCOMオブジェクトを使用して、ロギング機構をどのように達成するかを実演したものである。音声開発者は、ロギングの関心レベルを示す属性「ｌｏｇ」を関連するSMLノードに付け加える。上記の例では、アプリケーション開発者は、単一のｂｉｎｄ指示文（ｄｉｒｅｃｔｉｖｅ）を使用することによって、ログ値が３より大きいかまたは等しいすべてのノードをログするように選択する。この例は、ダウンレベルブラウザとアップレベルブラウザの両方で使用できる。

【０２４９】この例は、ｓｍｅｘオブジェクトが認識ド＊

例２：着信呼のアドレスを読み取る

```
<input type="text" id="remote"/>
<input type="text" id="transfer"/>
<input type="text" id="local"/>
<input type="hidden" id="session_id"/>
...
<smex id="telephone" sent="start_listening">
    <param name="server">http://tel-svr/whatever</param>
    <bind targetElement="session_id" value="//sid"/>
    <bind targetElement="remote" value="//remote_addr"/>
    <bind targetElement="transfer"
value="//transfer_addr"/>
    <bind targetElement="local" value="//local_addr"/>
    ...
</smex>
```

【０２５１】この例は、受け取ったメッセージを処理するためにどのようにｂｉｎｄ指示文を使用するかを示したものである。この例では、着信呼に関するメッセージが、コンテンツがそれぞれ着信呼のリモート、転送、およびローカルアドレスを表す、サブ要素ｒｅｍｏｔｅ＿ａｄｄｒ、ｔｒａｎｓｆｅｒ＿ａｄｄｒ、およびｌｏｃａｌ＿ａｄｄｒを有すると想定している。

【０２５２】この例では、電話サーバと通信するために、ＨＴＴＰに基づく接続のないプロトコルが使用される。ここで電話サーバは、複数のブラウザインスタンスと通信するように設計されるため、各クライアントは、アプリケーションが開始されたときにサーバによって割り当てられた固有のｉｄでそれ自体を識別しなければならない。これは、この例では、サーバに「ｓｔａｒｔ＿ｌｉｓｔｅｎｉｎｇ」メッセージを送信することによって達成されている。この例では、他の技法（たとえばクライアント側のクッキー）もセッション状態を管理するのに使用できるが、Ｗｅｂサーバに返送して、アプリケーションの次のページに渡すことのできるセッションｉｄが隠しフィールドに格納される。ｒｅｃｏの場合と同様に、あらゆるプラットフォームメッセージに対してあらゆるｂｉｎｄ指示文が実行されるわけではない。上記の例は、着信呼がある場合に、固有のｉｄが受け取られるだけであることを暗に示しているものではない。

【０２５３】７．１　プロパティ

ｓｍｅｘオブジェクトは、読取り／書込みプロパティのみが初期値指定用の属性としての役割も果たせるような、以下のプロパティを有することができる。

・ｓｅｎｔ：読取り／書込みであって、プラットフォーム構成要素に送信されるメッセージに対応する文字列。ｓｅｎｔがｖａｌｕｅとして使用される場合は必ず、そのコンテンツがディスパッチ（ｄｉｓｐａｔｃｈ）されるドキュメントにプラットフォームメッセージを返送する責務に関して混乱することがない限り、同じプラットフォーム構成要素と通信する複数のｓｍｅｘオブジェクトをページに含めることができることを実証するものでもある。上記の例は、構成要素が、それぞれが独自のｓｍｅｘまたはｍｅｓｓａｇｅコンジットを有する複数のインターフェースを実施することが可能であることを暗に示すものである。同じ引数が、複数のポートを聴取しているＴＣＰサーバに適用される。

【０２５０】

れる。プロパティがｒｖａｌｕｅとして使用される場合、またはこのプロパティにｎｕｌｌオブジェクトが割り当てられる場合は無効である。

・ｒｅｃｅｉｖｅｄ：読取り専用であって、受け取ったメッセージを示すXML DOM Nodeデータ。メッセージは、次のｏｎＲｅｃｅｉｖｅイベントの送信準備ができるまで、ｒｖａｌｕｅとして使用可能となる。

・ｔｉｍｅｒ：読取り／書込みであって、タイムアウトイベントがトリガされるまでの時間を示す、ミリ秒単位の数。プロパティに正の値が割り当てられると、クロックは時間刻みを開始する。カウントダウンの進行中に、値を変更することができる。ゼロまたは負の値は、タイムアウトイベントをトリガせずにクロックを停止させる。デフォルトの値は０であり、タイムアウトがないことを意味する。

・ｓｔａｔｕｓ：読取り専用であって、オブジェクトの最近の状況を示す整数。可能な値は０、－１、および－２であって、それぞれ、正常、タイムアウト満了、ならびにプラットフォームとの通信が確立できないかまたは中断された、という意味である。プラットフォーム特有のエラーメッセージは、ｒｅｃｅｉｖｅｄプロパティを介して搬送されなければならない。エラーメッセージが首尾よく送達された場合、ｓｔａｔｕｓコードは０である。

【０２５４】７．２　イベント

オブジェクトには以下のイベントがある。

＊・ｏｎＲｅｃｅｉｖｅ：このイベントは、プラットフォームメッセージが到達したときに送信される。ｂｉｎｄ要素によって宣言されたなんらかの指示文がある場合、それらの指示文は、イベントが開始される前に第１に評価される。送信の前に、ｒｅｃｅｉｖｅｄプロパティが更新される。

・ｏｎＥｒｒｏｒ：このイベントは、タイムアウトが満了したとき、または通信リンクエラーに遭遇したときに送信される。このイベントが送信されると、ｓｔａｔｕｓプロパティは、前述のように対応するエラーコードで更新される。

【０２５５】７．３　子要素

要素の形態を想定する場合、ｓｍｅｘは以下の子要素を有することができる。

・ｂｉｎｄ：受け取ったメッセージ上で指示文が動作している場合を除き、ｒｅｃｏの場合と同様。

・ｐａｒａｍ：ｒｅｃｏの場合と同様に、ｓｍｅｘオブジェクトにプラットフォーム特有のパラメータを提供する。各ｐａｒａｍ要素には、「ｎａｍｅ」属性を使用して命名することが可能であり、ｐａｒａｍ要素のコンテンツがパラメータの値である。一実施形態では、要素は、名前スペースおよびXMLデータタイプ宣言用の標準XML属性を理解しなければならない。

【０２５６】７．４　他の注釈

ロギング関数用にＳＭＥＸを拡張する的確な方法の１つは、次のようになる。

＊

```
<smex id="logServer" ...> ... </smex>
<script> function logMesseage(logClass, message) {
    logServer.sent = logClass + "|" + message;
} </script>
```

【０２５７】これは、実際には、動作がカスタマイズ可能な（グローバル）関数を使用してオブジェクトを拡張するものである。上記の例では、ロギング関数は、ｉｄとメッセージとの間にフィールドセパレータ「｜」を挿入するようにプログラムされる。

※【０２５８】グローバル関数を好まない人の場合、ＥＣＭＡＳｃｒｉｐｔの「ｐｒｏｔｏｔｙｐｅ」プロパティを使用して、オブジェクトメソッドとして関数を付け加えることができる。たとえば、以下のようになる。

※

```
<smex id="logServer" onload="addFunction()"> ... </smex>
<script>
    function my_logMessage(logClass, message) {
        logServer.sent = logClass + "|" + message;
    }
    function addFunction() {
        logServer.prototype.logMessage =
my_logMessage;
    }
</script>
```

【０２５９】よりオブジェクト指向の強い方法での関数は、以下のように表すことができる。

```
logServer.logMessage(RECO_LOG_ERROR, "My
message");
```

【０２６０】すべての必要な機構はすでに標準で確立されているが、上記の例に示すように、拡張作業を実行するための作業がｓｍｅｘオブジェクト実施者からさらに要求されることに留意されたい。

【図面の簡単な説明】

【図１】コンピューティングデバイスの動作環境の第１

の実施形態を表す平面図である。

【図２】図１のコンピューティングデバイスの構成図である。

【図３】電話の平面図である。

【図４】汎用コンピュータの構成図である。

【図５】クライアント／サーバシステム用のアーキテクチャを示す構成図である。

【図６】クレジットカード情報を取得するための画面を示す図である。

【図７】クライアント側で実行可能なマークアップ言語のページを示す図である。

【図８】表示および音声認識機能を有する、クライアント側で実行可能なマークアップ言語のページ例を示す図である。

【図９】可聴提示専用および主導権がシステムにある、クライアント側で実行可能なマークアップ言語のページ例を示す図である。

【図１０】可聴提示専用および主導権がシステムにある、クライアント側で実行可能なマークアップ言語のページ例を示す図である。

【図１１】可聴提示専用および主導権が混合の、クライアント側で実行可能なマークアップ言語のページ例を示す図である。

【図１２】可聴提示専用および主導権が混合の、クライアント側で実行可能なマークアップ言語のページ例を示す図である。

【図１３】サーバ側プラグインモジュールによって実行

可能なスクリプトの例を示す図である。

【図１４】認証サーバの第１の動作モードを示す絵画図である。

【図１５】認証サーバの第２の動作モードを示す絵画図である。

【図１６】認証サーバの第３の動作モードを示す絵画図である。

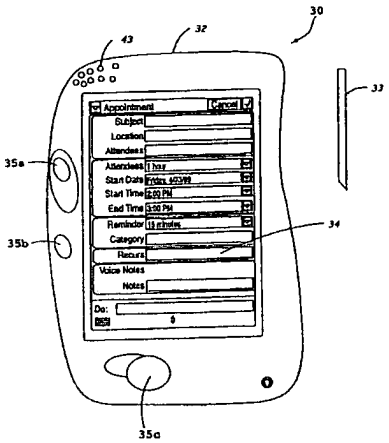
【図１７】スクリプトなしでクライアント側で実行可能な宣言型マークアップ言語のページ例を示す図である。

【図１８】スクリプトなしでクライアント側で実行可能な宣言型マークアップ言語のページ例を示す図である。

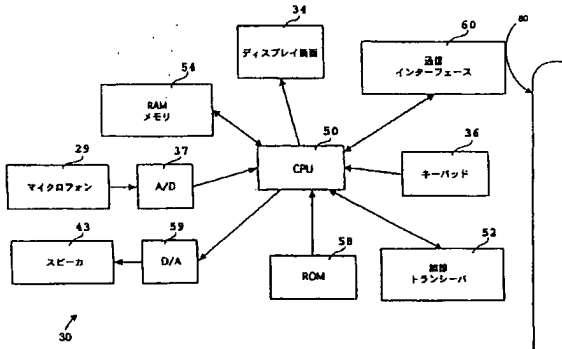
【符号の説明】

８０　単純な電話
２０２　Ｗｅｂサーバ
２０４　音声サーバ
２０５　インターネット
２０８　ＰＳＴＮ
２１０　第三者ＶｏＩＰ
２１１　音声などのＲｅｃｏ
２１２　電話音声ブラウザ
２１４　メディアサーバ（リモート可聴オブジェクト）
２１６　ＩＥ音声ブラウザ
３０７　ＴＴＳ
３０９　構文解析装置
３２０　ＡＳＰ＋ページ音声可能制御／テンプレート
３２４　ダイアログ制御

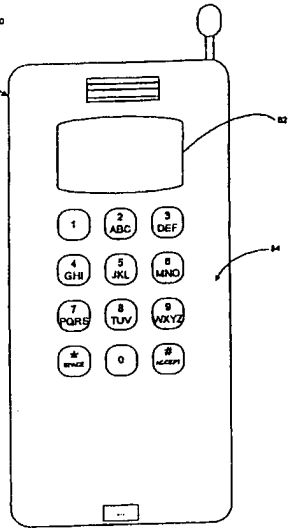
【図１】



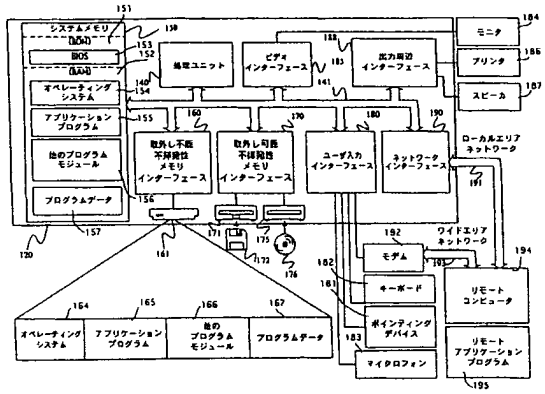
【図２】



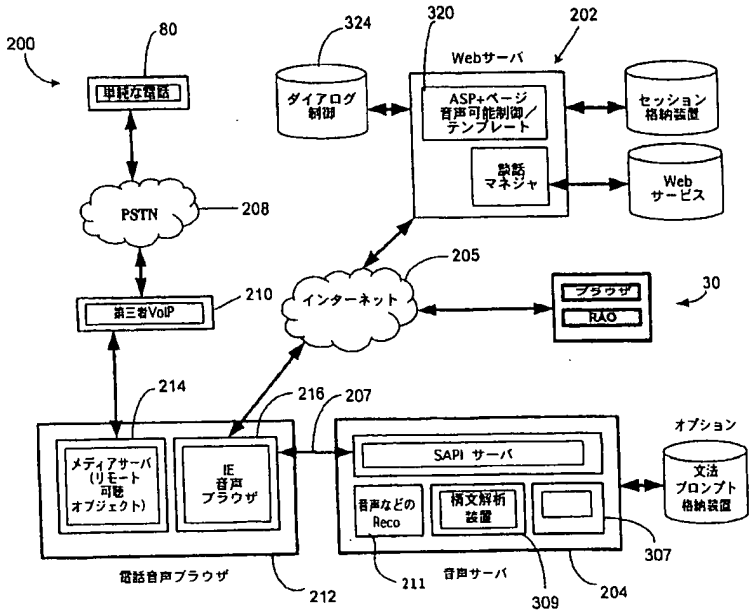
【図３】



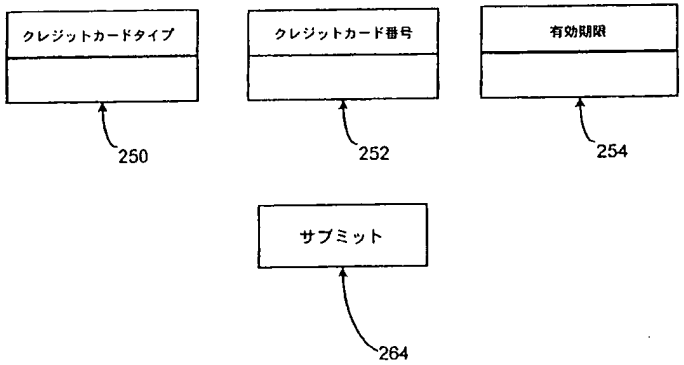
【図４】



【図５】



【図６】



【図７】

260

```
<html>
  <form id="get_card_info" method="post" action="http://payment.asp">
    <select name="card_type">
       <option value="amex">American Express</option>
       <option value="visa">Visa</option>
       <option value="ms">MasterCard</option>
    </select>
    <input type="text" name="card_num" width="30"
      onChange="handle()" />
    <input type="text" name="expiry_date" />
    <input type="submit" value="Submit" onClick="verify()" />
  </form>
  <script><![CDATA[
          function handle() {
            if (get_card_info.card_type.value == "amex") {
                if (get_card_info.card num.length != 15)
                   alert ("amex should have 15 digits");
            } else
                if (get_card_info.card_num.length != 16)
                   alert ("visa and master should have 16 digits");
          }
          function verify() {
            var flag = window.confirm("submit the credit card info?");
            if (flag)
                get_card_info.submit();
          }
    ]]>
  </script>
</html>
```

262

【図１０】

```
function _handleO {
  handle();
  checkFilled();
}
function checkFilled() {
  if (card_type.value == "") {
        p_card_type.active();
        g_card_types.activateO; return;
  }
  if (card_num.value =="") {
        p_card_num.active(); g_card_num.acti   vate(); return;
  }
  if (expiry_date.value == "") {
        p_expiry_data.active(); g_expiry_date.activate(); return;
  }
    p_content.activate();
  p_confirm.activate();
  confirmation.activateO;
}
  function confirmed(gobj) {
        if (gobj.recogRes.text == "yes")
              get_card_info.submit();
  }
  //
  // user codes start here
  //
  function handle() {

    if (get_card_info.card_type == "amex") {
      if (get_card_info.card_num.length != 15) {
        prompt.speak ("amex should have 15 digits");
        get_card_info.card_num = "";
      }
    } else
      if (get_card_info.card_num.length != 16)  {
        prompt.speak ('visa and master should have 16 digits");
        get_card_info.card_num = "";
      }
  }
]]>
</script>
</body>
```

【図８】

```
<html>
  <form id="get_card_info" method="post" action="http://payment.asp">
    <select name="card_type" onClick="talk(g_card_types)">
      <option value="amex">American Express</option>
      <option value="visa">Visa</option>
      <option value="ms">MasterCard</option>
    </select>
    <reco id="g_card_types" onReco ="handle()" >
      <grammar  src="./gram#card_types" />
    </reco >
    <input type="text" name="card_num" width="30"
      onClick="talk(g_card_num)" />
    <reco id=" g_card_num " onReco ="handle()" >
      <grammar  src=" ./gram#digits" />
    </reco >
    <input type="text" name="expiry_date"
      onClick="talk(g_expiry_date)" />
    <reco id="g_expiry_date " >
      <grammar  src ="  ./gram#dates " />
    </reco >
    <input type="submit" value="Submit" onClick="verify()" />
  </form>
  <script><![CDATA[
        function talk(gobj) {
            gobj.activate();
        }
        function handle() {
            if (get_card_info.card_num != null)  {
                if (get_card_info.card_type.value == "amex") {
                    if (get_card_info.card_num.length != 15)
                      alert ("amex should have 15 digits");
                } else
                    if (get_card_info.card_num.length != 16)
                      alert ("visa and master should have 16 digits");
            }
        }
        function verify() {
            var flag = window.confirm("submit the credit card
    info?");
            if (flag) {
                get_card_info.submit();
            }
        }
    ]]>
  </script>
</html>
```

270, 272, 290, 280, 282, 284

【図９】

```
< body >
  < form id =" get_card_info " method =" post" action =" http://payment.asp "
      onActivate =" welcom() ">
    < prompt id =" p_welcome ">  We now need your credit card  </ prompt >
    < prompt id =" p_mumble ">  I didn't understand you  </ prompt >
    < prompt id =" p _card_type " bargein =" true ">  What credit card would you
use? </ prompt >
    < prompt id =" p_card_num " bargein =" true ">  Please say the number"  </ prompt >
    < prompt id =" p_expiry_date " bargein =" true ">  What is the expiration
date? </ prompt >
    < prompt id =" p_content ">
        I have your  < value  select =" card_type " />  < value  select =" card_num " />
        with expiration date
      < value  select =" expiry_date " />
    </ prompt >
    < prompt id =" p_confirm ">  Is this correct? </ prompt >
    < reco id =" g_card_types "      onNoReco  =" mumble(this, 2) " >
     onReco  =" _handle(this, card_type)  " />
    < grammar  src =" ./gram#card_types "
     </reco >
    < reco id =" g_card_num " onNoReco  =" mumble(this, 1) " >
     onReco  =" _handle(this,  card_num) " />
      < grammar src =" ./gram#digits " />
     </reco >
    < reco id =" g_expiry_date " onNoReco  =" mumble(this, 1) " >
     onReco  =" _handle(this, expiry_date)  " audio =" rec " />
      < grammar src =" ./gram#dates " />
    </reco >
    < reco id =" confirmation "    onReco      =" confirmed(this) " />
      < grammar src =" ./gram#yesno " />
    </reco >
    < select name =" card_type ">
        < option value =" amex "> American Express  </ option >
        < option value =" visa "> Visa </ option >
        < option value =" ms "> MasterCard </ option >
      </ select >
      < input type =" text" name =" card_num " width =" 30 " />
      < input type =" text" name =" expiry_ date " />
      < input type =" submit" value =" Submit " />
  </ form >
< script > <![CDATA[
        function welcome() {
          p_welcome.active();
          repeat = 0;
          checkFilled();
        }
        function mumble(gobj, maxprompts) {
          gobj.deactivate();
          p_mumble.active();
          checkFilled();
        }
```

300, 303, 305, 302

図10へ

【図１１】

```
<body>
    <form id="get_card_info" method="post" action="http://payment.asp"
      onactivate="welcome()">
      <prompt id="p_welcome">We now need your credit card</prompt>
      <prompt id="p_mumble">I didn't understand you</prompt>
      <prompt id="p_card_type" bargein="true">What credit card would you
use?</prompt>
      <prompt id="p_card_num" bargein="true">Please say the number</prompt>
      <prompt id="p_expiry_date" bargein="true">What is the expiration
date?</prompt>
      <prompt id="p_content">
       I have your  <value select="card_type" /><value select="card_num" />
  with expiration date <value select="expiry_date" />
      </prompt>
      <prompt id="p_confirm">Is this correct?</prompt>
      <reco id="g_get_card_info" onReco="_handle()"  onNoReco ="mumble(this)">
        <grammar    src   ="./gram/getPayment" />
        <bind target="card_type" value="/card/type" />
        <bind target="card_num" value="/card/number" />
405 ——  <bind target="expiry_date" value="/card/expr_date" />
      </reco>
      <reco id="confirmation" onReco ="confirmed(this)  onNoReco="mumble(this)" />
        <grammar src ="./gram#yesno"/>
      </reco >
      <select name="card_type">
        <option value="amex">American Express</option>
        <option value="visa">Visa</option>
        <option value="ms">MasterCard</option>
      </select>
      <input type="text" name="card_num" width="30" />
      <input type="text" name="expiry_date" />
      <input type="submit" value="Submit" />
    </form>
    <script><![CDATA[
        function welcome() {
          p_welcome.active();
          repeat = 0;
          checkFilled();
        }
        function mumble(gobj) {
          gobj.deactivate();
          p_mumble.active();
          checkFilled();
        }
        function _handle() {
          handle();
          checkFilled();
        }
```

図12へ

【図１２】

```
        function checkFilled()  {
          if (card_type.value == "") {
                p_card_type.active();; do_field.activate(); return;
          }
          if (card_num.value =="") {
                p_card_num.active(); do_field.activate(); return;
          }
          if (expiry_date.value == "") {
                p_expiry_data.active(); do_field.activate(); return;
          }
          p_content.activate();
          p_confirm.activate();
          confirmation.activate();
        }
        function confirmed(gobj) {
          if (gobj.recogRes.text == "yes")
                get_card_info.submit(genSml());
        }
        //
        // user codes start here
        //
          function handle() {
          if (field == get_card_info.card_num) {
              if (get_card_info.card_num.length != 15) {
                  prompt.speak ("amex should have 15 digits");
                  get_card_info.card_num = "";
              }
          } else
              if (get_card_info.card_num.length != 16)  {
                  prompt.speak ("visa should have 15 digits");
                  get_card_info.card_num = "";
              }
        }
          function genSml() {
                str = '<sml><credit_card type="';
                str += card_type.value; str += '"><number>';
                str += card_number.value; str += '</number><expire>';
                str += expiry_date.value; str +=
'</expire></credit_card></sml>';
                return str;
          }
    ]]>
  </script>
</body>
```

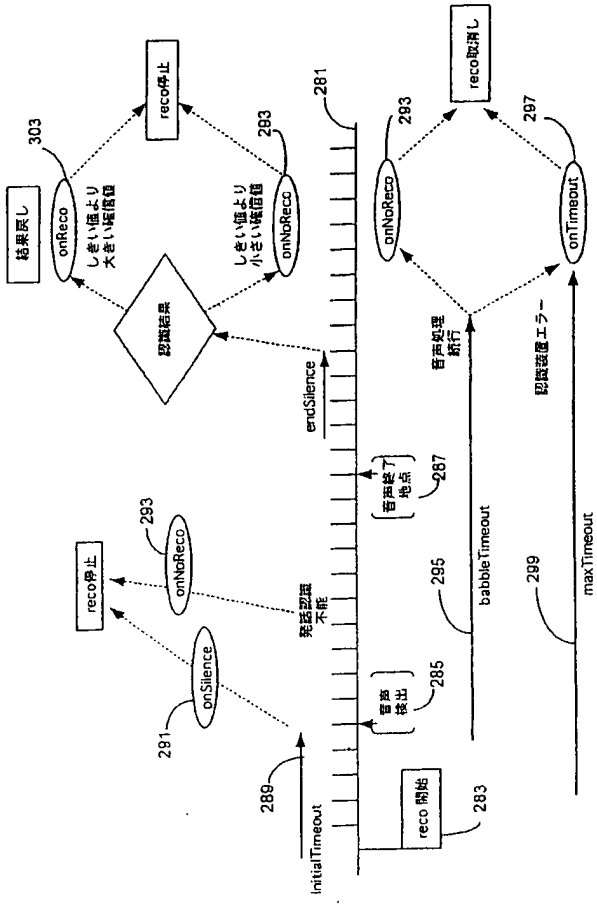
【図１３】

SAMPLE ASP+ PAGE

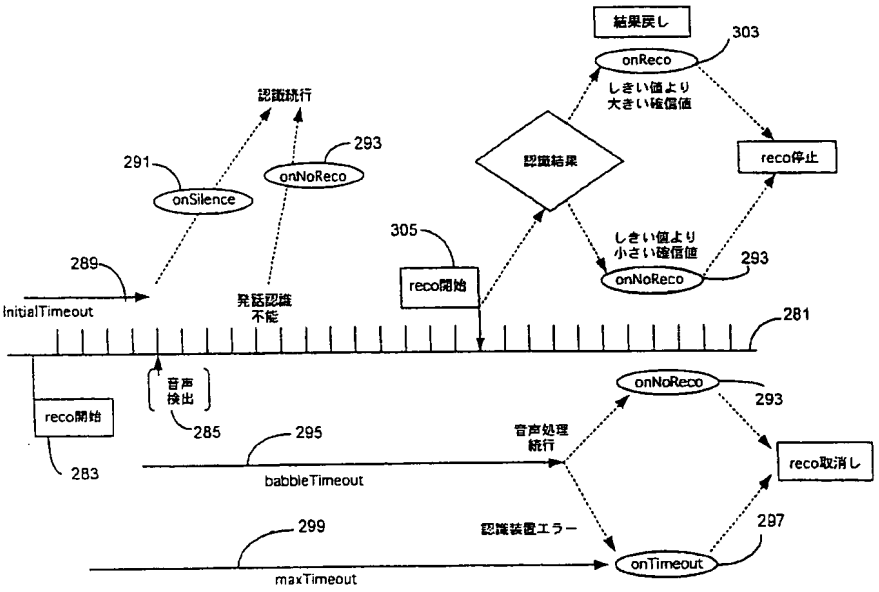
```
<%@ Page language="Jscript" AutoEventWireup="false" Inherits="Credit.Transaction" %>
<html><head>
<!-- ASPX page for both voice-only & multimodal credit card example -->
<script>
function handle() {
        if (field == get_card_info.card_num) {
           if (get_card_info.card_num.length != 15) {
            prompt.speak ("amex should have 15 digits");
            get_card_info.card_num = "";
           }
        } else
           if (get_card_info.card_num.length != 16)  {
            prompt.speak ("visa should have 15 digits");
            get_card_info.card_num = "";
           }
      }
function gensml() {
      str = '<sml><credit_card><card_type>';
      str += card_type.value; str += '</card_type><number>';
      str += card_number.value; str += '</number><expire>';
      str += expiry_date.value; str += '</expire></credit_card></sml>';
      return str;
}
</script>
<script runat="server">
function Page_Load (obj, args) {
      if (PostBack) {
        validator = new System.Speech.SMLValidator("./CreditSDL.xml");
        dsml = validator.Evaluate(args);
        Navigate(ChoosePage(dsml));
      } else {
        // initialize fields with args
      }
}
</script>
</head>
<body>
<speech:form id="get_card_info" style="system_initiative"
      prompt="./prompt/getPayment" onsubmit="gensml()">
  <speech:choice name="card_type" prompt="What credit card would you use?"
      grammar="./gram#card_types" onPhraseFinish="handle()"
      <option>American Express</option>
      <option>Visa</option>
      <option>Mastercard </option>
  </speech:choice>
  <speech:textbox name="card_number" prompt="Please say the number"
      grammar="./gram#digits" onPhraseFinish="handle()">
  <speech:textbox name="expiry_date" prompt="What is the expiration date? "
      grammar="./gram#dates" onPhraseFinish="handle()"/>

</speech:form>
</body>
</html>
```

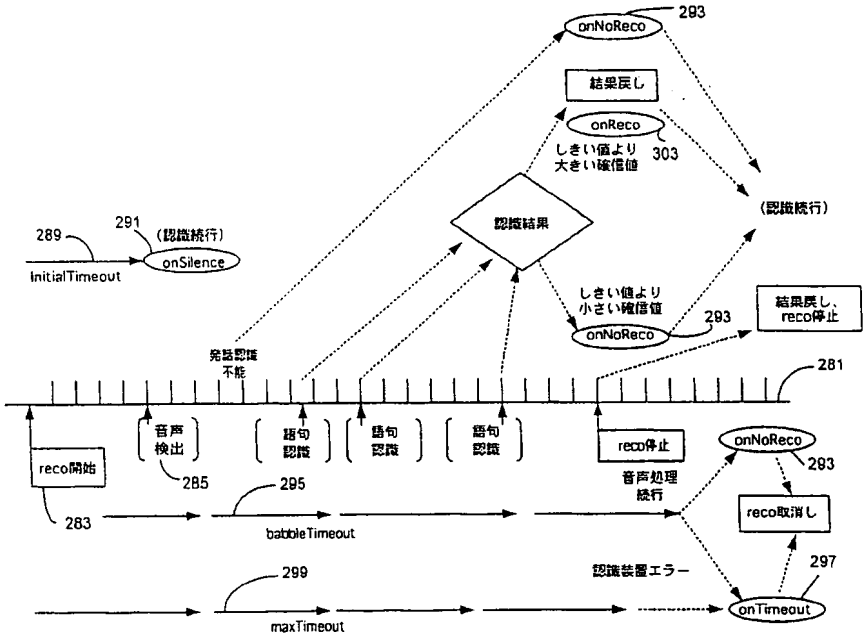

【図１４】



【図15】


結果戻し
303
onReco
しきい値より
大きい確信値
認識結果
reco停止
認識続行
291
293
onSilence
onNoReco
305
しきい値より
小さい確信値
293
289
reco開始
onNoReco
InitialTimeout
発話認識
不能
281
音声
検出
onNoReco
293
reco開始
285
295
音声処理
続行
reco取消し
283
babbleTimeout
299
認識装置エラー
297
onTimeout
maxTimeout

【図16】


onNoReco
293
結果戻し
onReco
しきい値より
大きい確信値
303
(認識続行)
289
291
(認識続行)
認識結果
onSilence
InitialTimeout
しきい値より
小さい確信値
結果戻し、
reco停止
293
onNoReco
発話認識
不能
281
音声
検出
語句
認識
語句
認識
語句
認識
reco停止
onNoReco
293
reco開始
285
音声処理
続行
295
reco取消し
283
babbleTimeout
299
認識装置エラー
297
onTimeout
maxTimeout

【図１７】

```
<html>
<body>
<!-- the data section -->
  <form id="get_drink">
    <input name="drink" />
    <input type="radio" name="cream"/>
    <input type="radio" name="sugar"/>
    <input name="uid" type="hidden"/>
  </form>
<!-- The speech section -->
<prompt id="welcome">Welcome, caller! </prompt>
<prompt id="ask"> Do you want coke, coffee, or orange juice?
</prompt>
<prompt id="confirm"> I heard <value href="drink"/>. Is this
    correct? </prompt>
<prompt id="thanks"> Thank you. Please wait when I get it for
    you. </prompt>
<prompt id="retry"> Okay, let's do this again </prompt>
<prompt id="reprompt"> Sorry, I missed that. </prompt>
<prompt id="cream_sugar">Do you want cream or sugar with your
    coffee?</prompt>

<reco id ="reco_drink"><grammar src="./drinktypes"/>
  <bind test="/[@confidence $le$ 10]"
    targetElement="reprompt" targetMethod="start"
    targetElement="ask" targetMethod="start"
    targetElement="reco_drink" targetMethod="start"/>
  <bind test="/drink/coffee[@confidence $gt$ 10]"
    targetElement="drink" value="/drink"
    targetElement="cream_sugar" targetMethod="start"
    targetElement="reco_cream_sugar"
        targetMethod="start"/>
    <bind test="/[@confidence $gt$ 10]"
      targetElement="drink" value="/drink"
      targetElement="confirm" targetMethod="start"
      targetElement="reco_yesno" targetMethod="start"/>
  </reco>
```

350, 352, 354, 364, 366, 368, 370, 372, 374, 376, 378, 382

【図１８】

```
<reco id="reco_cream_sugar"><grammar src="./cream+sugar"/>
    <bind test="/[@confidence $gt$ 10 and
        host()/get_drink/drink = 'coffee']"
        targetElement="cream" targetAttribute="checked"
            value="/cream/@value"
        targetElement="sugar" targetAttribute="checked"
            value="/sugar/@value"
        targetElement="confirm" targetMethod="start"
        targetElement="reco_yesno" targetMethod="start"/>
</reco>

<reco id="reco_yesno"> <grammar src="./yesno"/>
        <bind test="/yes[@confidence $gt$ 10]"
    384   targetElement="thanks" targetMethod="start"
    386   targetElement="get_drink"
            targetMethod="submit" />
        <bind test="/no or ./[@confidence $le$ 10]"
    390   targetElement="retry" targetMethod="start"
    392   targetElement="ask" targetMethod="start"
    394   targetElement="reco_drink"
            targetMethod="start"/>
</reco>

<!-call control section -->
<smex id="telephone" sent="start_listening"><param
server="ccxmlproc"> … </param>
        <bind targetElement="uid" value="/@uid"/>
        <bind test="/Call_connected"
    361   targetElement="welcome" targetMethod="start"
    362   targetElement="ask" targetMethod="start"
    363   targetElement="reco_drink"
            targetMethod="start"/>
</smex>
</body>
</html>
```

356, 358, 360

フロントページの続き

(72)発明者　ウァン　クァンサン
アメリカ合衆国　98006　ワシントン州　ベルビュー　サウスイースト　48　コート　16470

(72)発明者　ホン　シャオーウェン
アメリカ合衆国　98006　ワシントン州　ベルビュー　サウスイースト　58　プレイス　17797

Ｆターム(参考)　5D015 KK03
5E501 AA03 AA04 AB03 BA05 BA11 CB04 CB05 CB06 CB08 CB09 CB10 CB13 CB15 CC17 EA21 FA04

【外国語明細書】

1. Title of the Invention

MARKUP LANGUAGE EXTENSIONS FOR WEB ENABLED RECOGNITION

2. Claims

1. A computer readable medium having a markup language for execution on a client device in a client/server system, the markup language comprising an instruction indicating a grammar to associate with input entered through the client device.

2. The computer readable medium of claim 1 wherein the instruction indicating a grammar provides a reference to a location of the grammar.

3. The computer readable medium of claim 1 wherein the grammar is for one of speech recognition, handwriting recognition, gesture recognition and visual recognition.

4. The computer readable medium of claim 3 wherein the instruction indicating a grammar includes a reference to a grammar for speech recognition.

5. The computer readable medium of claim 1 wherein the markup language includes an instruction for receiving a recognition result pertaining to recognized speech and associating the result with a data field on the client device.

6. The computer readable medium of claim 8 wherein the instruction for receiving a recognition result associates the recognition result to a plurality of data fields.

7. The computer readable medium of claim 1 wherein the markup language includes an instruction to audibly render an indication when the speech was not recognized.

8. The computer readable medium of claim 1 wherein the input comprises a text string.

9. The computer readable medium of claim 8 wherein the markup language includes an instruction for receiving the text string from another component.

10. The computer readable medium of claim 9 wherein markup language associates the grammar to process the text string.

11. A computer readable medium including instructions readable by a computer which, when implemented, cause the computer to handle information by performing steps comprising:

receiving a markup language page from a web server having a field for input data;

receiving input from a user related to the field; and

sending data indicative of the input and an indication of a grammar for recognition.

12. The computer readable medium of claim 11 wherein the indication provides a reference to locations of the grammars.

13. The computer readable medium of claim 12 wherein the indication includes a reference to a language for speech recognition.

14. A method for recognition in a client/server network, the method comprising:

receiving a markup language page having a field for input data on a client device connected to a network from a web server at an address on the network different than the client device;

receiving input from a user related to the field on the client device; and

sending data indicative of the input and an indication of a grammar for recognition to a recognition server at an address on the network different than the client device.

15. The method of claim 14 and further comprising:

normalizing the data indicative of the input prior to sending the data.

16. The method of claim 41 and further comprising:

receiving a recognition result pertaining to recognized speech and associating the result with a data field on the client device.

17. A computer readable medium having a markup language for execution on a client device in a client/server system, the markup language comprising an instruction indicating an object model element having an attribute or a method for recognition.

18. The computer readable medium of claim 17 wherein the element is for one of speech recognition, handwriting recognition, gesture recognition, DTMF recognition and visual recognition.

19. The computer readable medium of claim 17 wherein the element comprises an indication of a grammar for recognition.

20. The computer readable medium of claim 19 wherein element pertains to receiving and processing a text string like recognition.

21. The computer readable medium of claim 20 and further comprising a second object model receiving the text string from another component.

22. A computer readable medium having a markup language for execution on a client device in a client/server system, the markup language comprising an instruction indicating an object model element having an attribute or a method for audibly prompting.

23. The computer readable medium of claim 22 wherein the markup language comprises one of HTML, XHTML, cHTML, XML and WML.

24. The computer readable medium of claim 22 wherein the markup language comprises a scripting language.

25. The computer readable medium of claim 22 wherein the markup language comprises a synchronized markup language.

3. Detailed Description of the Invention

FIELD OF THE INVENTION

The present invention relates to access of information over a wide area network such as the Internet. More particularly, the present invention relates to web enabled recognition allowing information and control on a client side to be entered using a variety of methods.

BACKGROUND OF THE INVENTION

Small computing devices such as personal information managers (PIM), devices and portable phones are used with ever increasing frequency by people in their day-to-day activities. With the increase in processing power now available for microprocessors used to run these devices, the functionality of these devices are increasing, and in some cases, merging. For instance, many portable phones now can be used to access and browse the Internet as well as can be used to store personal information such as addresses, phone numbers and the like.

In view that these computing devices are being used for browsing the Internet, or are used in other server/client architectures, it is therefore necessary to enter information into the computing device. Unfortunately, due to the desire to keep these devices as small as possible in order that they are easily carried, conventional keyboards having all the letters of the alphabet as isolated buttons are usually not possible due to the limited surface area available on the housings of the computing devices.

Recently, voice portals such as through the use of VoiceXML (voice extensible markup language) have been advanced to allow Internet content to be accessed using only a telephone. In this architecture, a document server (for

example, a web server) processes requests from a client through a VoiceXML interpreter. The web server can produce VoiceXML documents in reply, which are processed by the VoiceXML interpreter and rendered audibly to the user. Using voice commands through voice recognition, the user can navigate the web.

VoiceXML is a markup language with flow control tags; however, flow control does not follow the HTML (Hyper Text Markup Language) flow control model, which includes eventing and separate scripts. Rather, VoiceXML generally includes a form interpretation algorithm that is particularly suited for telephone-based voice-only interaction, and commonly, where the information obtained from the user is under the control of the system or application. Incorporation of VoiceXML directly into applications available in a client-server relationship where graphically user interfaces are also provided will require the developer to master two forms of web authoring, one for VoiceXML and the other using HTML (or the like), each one following a different flow control model.

There is thus an ongoing need to improve upon the architecture and methods used to provide speech recognition in a server/client architecture such as the Internet. The authoring tool for speech recognition should be easily adaptable to small computing devices such as PIMs, telephones and the like. An architecture or method of web authoring that addresses one, several or all of the foregoing disadvantages is particularly needed.

SUMMARY OF THE INVENTION

A markup language for execution on a client device in a client/server system includes an instruction indicating a grammar to associate with input data entered through the client device.

With the availability of this extension and as another aspect of the present invention, a client device can execute instructions to receive a markup language page from a web server having a field for input data. The client device can then receive input data from a user related to the field and send the data and an indication of the grammar for recognition to a recognition server, typically, located at a remote location for processing.

The recognition server can execute instructions to receive the input data and the indication of the grammar to perform recognition. The results of recognition can then be sent back to the client device or web server for further processing.

DETAILED DESCRIPTION OF THE INVENTION

Before describing an architecture of web based recognition and methods for implementing the same, it may be useful to describe generally computing devices that can function in the architecture. Referring now to FIG. 1, an exemplary form of a data management device (PIM, PDA or the like) is illustrated at 30. However, it is contemplated that the present invention can also be practiced using other computing devices discussed below, and in particular, those computing devices having limited surface areas for input buttons or the like. For example, phones and/or data management devices will also benefit from the present invention. Such devices will have an enhanced utility compared to existing portable personal information management devices and other portable electronic devices, and the functions and compact size of such devices will more likely encourage the user to carry the device at all times. Accordingly, it is not intended that the scope of the architecture herein described be limited by the disclosure of

an exemplary data management or PIM device, phone or computer herein illustrated.

An exemplary form of a data management mobile device 30 is illustrated in FIG. 1. The mobile device 30 includes a housing 32 and has an user interface including a display 34, which uses a contact sensitive display screen in conjunction with a stylus 33. The stylus 33 is used to press or contact the display 34 at designated coordinates to select a field, to selectively move a starting position of a cursor, or to otherwise provide command information such as through gestures or handwriting. Alternatively, or in addition, one or more buttons 35 can be included on the device 30 for navigation. In addition, other input mechanisms such as rotatable wheels, rollers or the like can also be provided. However, it should be noted that the invention is not intended to be limited by these forms of input mechanisms. For instance, another form of input can include a visual input such as through computer vision.

Referring now to FIG. 2, a block diagram illustrates the functional components comprising the mobile device 30. A central processing unit (CPU) 50 implements the software control functions. CPU 50 is coupled to display 34 so that text and graphic icons generated in accordance with the controlling software appear on the display 34. A speaker 43 can be coupled to CPU 50 typically with a digital-to-analog converter 59 to provide an audible output. Data that is downloaded or entered by the user into the mobile device 30 is stored in a non-volatile read/write random access memory store 54 bi-directionally coupled to the CPU 50. Random access memory (RAM) 54 provides volatile storage for instructions that are executed by CPU 50, and storage for temporary data, such as register values. Default values for

configuration options and other variables are stored in a read only memory (ROM) 58. ROM 58 can also be used to store the operating system software for the device that controls the basic functionality of the mobile 30 and other operating system kernel functions (e.g., the loading of software components into RAM 54).

RAM 54 also serves as a storage for the code in the manner analogous to the function of a hard drive on a PC that is used to store application programs. It should be noted that although non-volatile memory is used for storing the code, it alternatively can be stored in volatile memory that is not used for execution of the code.

Wireless signals can be transmitted/received by the mobile device through a wireless transceiver 52, which is coupled to CPU 50. An optional communication interface 60 can also be provided for downloading data directly from a computer (e.g., desktop computer), or from a wired network, if desired. Accordingly, interface 60 can comprise various forms of communication devices, for example, an infrared link, modem, a network card, or the like.

Mobile device 30 includes a microphone 29, and analog-to-digital (A/D) converter 37, and an optional recognition program (speech, DTMF, handwriting, gesture or computer vision) stored in store 54. By way of example, in response to audible information, instructions or commands from a user of device 30, microphone 29 provides speech signals, which are digitized by A/D converter 37. The speech recognition program can perform normalization and/or feature extraction functions on the digitized speech signals to obtain intermediate speech recognition results. Using wireless transceiver 52 or communication interface 60, speech data is transmitted to a remote recognition server 204 discussed below and illustrated

in the architecture of FIG. 5. Recognition results are then returned to mobile device 30 for rendering (e.g. visual and/or audible) thereon, and eventual transmission to a web server 202 (FIG. 5), wherein the web server 202 and mobile device 30 operate in a client/server relationship. Similar processing can be used for other forms of input. For example, handwriting input can be digitized with or without pre-processing on device 30. Like the speech data, this form of input can be transmitted to the recognition server 204 for recognition wherein the recognition results are returned to at least one of the device 30 and/or web server 202. Likewise, DTMF data, gesture data and visual data can be processed similarly. Depending on the form of input, device 30 (and the other forms of clients discussed below) would include necessary hardware such as a camera for visual input.

FIG. 3 is a plan view of an exemplary embodiment of a portable phone 80. The phone 80 includes a display 82 and a keypad 84. Generally, the block diagram of FIG. 2 applies to the phone of FIG. 3, although additional circuitry necessary to perform other functions may be required. For instance, a transceiver necessary to operate as a phone will be required for the embodiment of FIG. 2; however, such circuitry is not pertinent to the present invention.

In addition to the portable or mobile computing devices described above, it should also be understood that the present invention can be used with numerous other computing devices such as a general desktop computer. For instance, the present invention will allow a user with limited physical abilities to input or enter text into a computer or other computing device when other conventional input devices, such as a full alpha-numeric keyboard, are too difficult to operate.

The invention is also operational with numerous other general purpose or special purpose computing systems, environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, regular telephones (without any screen) personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

The following is a brief description of a general purpose computer 120 illustrated in FIG. 4. However, the computer 120 is again only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computer 120 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated therein.

The invention may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote computer storage media including memory storage devices. Tasks performed by the

programs and modules are described below and with the aid of figures. Those skilled in the art can implement the description and figures as processor executable instructions, which can be written on any form of a computer readable medium.

With reference to FIG. 4, components of computer 120 may include, but are not limited to, a processing unit 140, a system memory 150, and a system bus 141 that couples various system components including the system memory to the processing unit 140. The system bus 141 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Universal Serial Bus (USB), Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus. Computer 120 typically includes a variety of computer readable mediums. Computer readable mediums can be any available media that can be accessed by computer 120 and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer readable mediums may comprise computer storage media and communication media. Computer storage media includes both volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital

versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computer 120.

Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, FR, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

The system memory 150 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 151 and random access memory (RAM) 152. A basic input/output system 153 (BIOS), containing the basic routines that help to transfer information between elements within computer 120, such as during start-up, is typically stored in ROM 151. RAM 152 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 140. By way of example, and not limitation, FIG. 4 illustrates operating system 54, application programs 155, other program modules 156, and program data 157.

The computer 120 may also include other removable/non-removable volatile/nonvolatile computer storage media. By way of example only, FIG. 4 illustrates a hard disk drive 161 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 171 that reads from or writes to a removable, nonvolatile magnetic disk 172, and an optical disk drive 175 that reads from or writes to a removable, nonvolatile optical disk 176 such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 161 is typically connected to the system bus 141 through a non-removable memory interface such as interface 160, and magnetic disk drive 171 and optical disk drive 175 are typically connected to the system bus 141 by a removable memory interface, such as interface 170.

The drives and their associated computer storage media discussed above and illustrated in FIG. 4, provide storage of computer readable instructions, data structures, program modules and other data for the computer 120. In FIG. 4, for example, hard disk drive 161 is illustrated as storing operating system 164, application programs 165, other program modules 166, and program data 167. Note that these components can either be the same as or different from operating system 154, application programs 155, other program modules 156, and program data 157. Operating system 164, application programs 165, other program modules 166,

and program data 167 are given different numbers here to illustrate that, at a minimum, they are different copies.

A user may enter commands and information into the computer 120 through input devices such as a keyboard 182, a microphone 183, and a pointing device 181, such as a mouse, trackball or touch pad. Other input devices (not shown) may include a joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 140 through a user input interface 180 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 184 or other type of display device is also connected to the system bus 141 via an interface, such as a video interface 185. In addition to the monitor, computers may also include other peripheral output devices such as speakers 187 and printer 186, which may be connected through an output peripheral interface 188.

The computer 120 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 194. The remote computer 194 may be a personal computer, a hand-held device, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 120. The logical connections depicted in FIG. 4 include a local area network (LAN) 191 and a wide area network (WAN) 193, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the computer 120 is connected to the LAN 191 through a network interface or adapter 190. When used in a WAN networking environment, the computer 120 typically includes a modem 192 or other means for establishing communications over the WAN 193, such as the Internet. The modem 192, which may be internal or external, may be connected to the system bus 141 via the user input interface 180, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 120, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. 4 illustrates remote application programs 195 as residing on remote computer 194. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

FIG. 5 illustrates architecture 200 for web based recognition as can be embodied in the present invention. Generally, information stored in a web server 202 can be accessed through mobile device 30 (which herein also represents other forms of computing devices having a display screen, a microphone, a camera, a touch sensitive panel, etc., as required based on the form of input), or through phone 80 wherein information is requested audibly or through tones generated by phone 80 in response to keys depressed and wherein information from web server 202 is provided only audibly back to the user.

More importantly though, architecture 200 is unified in that whether information is obtained through device 30 or phone 80 using speech recognition, a single recognition server 204 can support either mode of operation. In addition, architecture 200 operates using an

extension of well-known mark-up languages (e.g. HTML, XHTML, cHTML, XML, WML, and the like). Thus, information stored on web server 202 can also be accessed using well-known GUI methods found in these mark-up languages. By using an extension of well-known mark-up languages, authoring on the web server 202 is easier, and legacy applications currently existing can be also easily modified to include voice recognition.

Generally, device 30 executes HTML pages, scripts, or the like, provided by web server 202. When voice recognition is required, by way of example, speech data, which can be digitized audio signals or speech features wherein the audio signals have been preprocessed by device 30 as discussed above, are provided to recognition server 204 with an indication of a grammar or language model to use during speech recognition. The implementation of the recognition server 204 can take many forms, one of which is illustrated, but generally includes a recognizer 211. The results of recognition are provided back to device 30 for local rendering if desired or appropriate. Upon compilation of information through recognition and any graphical user interface if used, device 30 sends the information to web server 202 for further processing and receipt of further HTML pages/scripts, if necessary.

As illustrated in FIG. 5, device 30, web server 202 and recognition server 204 are commonly connected, and separately addressable, through a network 205, herein a wide area network such as the Internet. It therefore is not necessary that any of these devices be physically located adjacent each other. In particular, it is not necessary that web server 202 includes recognition server 204. In

this manner, authoring at web server 202 can be focused on the application to which it is intended without the authors needing to know the intricacies of recognition server 204. Rather, recognition server 204 can be independently designed and connected to the network 205, and thereby, be updated and improved without further changes required at web server 202. As discussed below, web server 202 can also include an authoring mechanism that can dynamically generate client-side markups and scripts. In a further embodiment, the web server 202, recognition server 204 and client 30 may be combined depending on the capabilities of the implementing machines. For instance, if the client comprises a general purpose computer, e.g. a personal computer, the client may include the recognition server 204. Likewise, if desired, the web server 202 and recognition server 204 can be incorporated into a single machine.

An aspect of the present invention is a method for processing input data in a client/server system that includes receiving from a server a markup language page having extensions configured to obtain input data from a user of a client device; executing the markup language page on the client device; transmitting input data (indicative of speech, DTMF, handwriting, gestures or images obtained from the user) and an associated grammar to a recognition server remote from the client; and receiving a recognition result from the recognition server at the client. Another aspect is a computer readable medium having a markup language for execution on a client device in a client/server system, the markup language having an instruction indicating a grammar to associate with input data entered through the client device.

Access to web server 202 through phone 80 includes connection of phone 80 to a wired or wireless telephone network 208, that in turn, connects phone 80 to a third party gateway 210. Gateway 210 connects phone 80 to a telephony voice browser 212. Telephony voice browser 212 includes a media server 214 that provides a telephony interface and a voice browser 216. Like device 30, telephony voice browser 212 receives HTML pages/scripts or the like from web server 202. More importantly though, the HTML pages/scripts are of the form similar to HTML pages/scripts provided to device 30. In this manner, web server 202 need not support device 30 and phone 80 separately, or even support standard GUI clients separately. Rather, a common mark-up language can be used. In addition, like device 30, voice recognition from audible signals transmitted by phone 80 are provided from voice browser 216 to recognition server 204, either through the network 205, or through a dedicated line 207, for example, using TCP/IP. Web server 202, recognition server 204 and telephone voice browser 212 can be embodied in any suitable computing environment such as the general purpose desktop computer illustrated in FIG. 4.

However, it should be noted that if DTMF recognition is employed, this form of recognition would generally be performed at the media server 214, rather than at the recognition server 204. In other words, the DTMF grammar would be used by the media server.

As indicated above, one aspect of the present invention includes extension of mark-up languages such as HTML, XHTML cHTML, XML, WML or with any other SGML-derived markup to include controls and/or objects that provide recognition in a client/server architecture. In this

manner, authors can leverage all the tools and expertise in these mark-up languages that are the predominant web development platform used in such architectures.

Generally, controls and/or objects can include one or more of the following functions: recognizer controls and/or objects for recognizer configuration, recognizer execution and/or post-processing; synthesizer controls and/or objects for synthesizer configuration and prompt playing; grammar controls and/or objects for specifying input grammar resources; and/or binding controls and/or objects for processing recognition results. The extensions are designed to be a lightweight markup layer, which adds the power of an audible, visual, handwriting, etc. interface to existing markup languages. As such, the extensions can remain independent of: the high-level page in which they are contained, e.g. HTML; the low-level formats which the extensions used to refer to linguistic resources, e.g. the text-to-speech and grammar formats; and the individual properties of the recognition and speech synthesis platforms used in the recognition server 204.

Before describing mark-up languages having controls and/or objects suited for recognition, it may be helpful to examine a simple GUI example herein embodied with the HTML mark-up language. Referring to FIG. 6, a simple GUI interface comprises submission of credit card information to the web server to complete an on-line sale. In this example, the credit card information includes a field 250 for entry of the type of credit card being used, for example, Visa, MasterCard or American Express. A second field 252 allows entry of the credit card number, while a third field 254 allows entry of the expiration date. Submit

button 264 is proviced to transmit the information entered in fields 250, 252 and 254.

FIG. 7 illustrates the HTML code for obtaining the foregoing credit card information from the client. Generally, as is common in these forms of mark-up languages, the code includes a body portion 260 and a script portion 262. The body portion 260 includes lines of code indicating the type of action to be performed, the form to use, the various fields of information 250, 252 and 254, as well as a code for submit button 264 (FIG. 6). This example also illustrates eventing support and embedded script hosting, wherein upon activation of the submit button 264, a function "verify" is called or executed in script portion 262. The "verify" function ascertains whether the card number length for each of the credit cards (Visa, MasterCard and American Express) is of the proper length.

FIG. 8 illustrates a client markup that generates the same GUI of FIG. 6 for obtaining credit card information to be provided to web server 204 using speech recognition. Although speech recognition will be discussed below with respect to FIGS. 8-14, it should be understood that the techniques described can be similarly applied in handwriting recognition, gesture recognition and image recognition.

Generally, the extensions (also commonly known as "tags") are a small set of XML elements, with associated attributes and DOM object properties, events and methods, which may be used in conjunction with a source markup document to apply a recognition interface, DTMF or call control to a source page. The extensions formalities and semantics are independent of the nature of the source

document, so the extensions can be used equally effectively within HTML, XHTML, cHTML, XML, WML, or with any other SGML-derived markup. The extension follow the document object model wherein new functional objects or elements, which can be hierarchical, are provided. Each of the elements are discussed in detail in the Appendix, but generally the elements can include attributes, properties, methods, events and/or other "child" elements.

At this point, it should also be noted that the extensions may be interpreted in two different "modes" according to the capabilities of the device upon which the browser is being executed on. In a first mode, "object mode", the full capabilities are available. The programmatic manipulation of the extensions by an application is performed by whatever mechanisms are enabled by the browser on the device, e.g. a JScript interpreter in an XHTML browser, or a WMLScript interpreter in a WML browser. For this reason, only a small set of core properties and methods of the extensions need to be defined, and these manipulated by whatever programmatic mechanisms exist on the device or client side. The object mode provides eventing and scripting and can offer greater functionality to give the dialog author a much finer client-side control over speech interactions. As used herein, a browser that supports full event and scripting is called an "uplevel browser". This form of a browser will support all the attributes, properties, methods and events of the extensions. Uplevel browsers are commonly found on devices with greater processing capabilities.

The extensions can also be supported in a "declarative mode". As used herein, a browser operating in a declarative mode is called a "downlevel browser" and does

not support full eventing and scripting capabilities. Rather, this form of browser will support the declarative aspects of a given extension (i.e. the core element and attributes), but possibly not all the DOM (document object model) object properties, methods and events. This mode employs exclusively declarative syntax, and may further be used in conjunction with declarative multimedia synchronization and coordination mechanisms (synchronized markup language) such as SMIL (Synchronized Multimedia Integration Language) 2.0. Downlevel browsers will typically be found on devices with limited processing capabilities.

At this point though, a particular mode of entry should be discussed. In particular, use of speech recognition in conjunction with at least a display and, in a further embodiment, a pointing device as well to indicate the fields for data entry is particularly useful. Specifically, in this mode of data entry, the user is generally under control of when to select a field and provide corresponding information. For instance, in the example of FIG. 6, a user could first decide to enter the credit card number in field 252 and then enter the type of credit card in field 250 followed by the expiration date in field 254. Likewise, the user could return back to field 252 and correct an errant entry, if desired. When combined with speech recognition as described below, an easy and natural form of navigation is provided. As used herein, this form of entry using both a screen display allowing free form selection of fields and voice recognition is called "multi-modal".

Referring back to FIG. 8, an example of HTML mark-up language code is illustrated. Like the HTML code

illustrated in FIG. 7, this code also includes a body portion 270 and a script portion 272. Also like the code illustrated in FIG. 7, the code illustrated in FIG. 8 includes indications as to the type of action to perform as well as the location of the form. Entry of information in each of the fields 250, 252 and 254 is controlled or executed by code portions 280, 282 and 284, respectively. Referring first to code portion 280, on selection of field 250, for example, by use of stylus 33 of device 30, the event "onClick" is initiated which calls or executes function "talk" in script portion 272. This action activates a grammar used for speech recognition that is associated with the type of data generally expected in field 250. This type of interaction, which involves more than one technique of input (e.g. voice and pen-click/roller) is referred as "multimodal".

It should be noted that the speech recognition extensions exemplified in Fig. 8 are not intended to have a default visual representation on the browser of the client, since for many applications it is assumed that the author will signal the speech enablement of the various components of the page by using application-specification graphical mechanisms in the source page. Nevertheless, if visual representations are desired, the extensions can so be modified.

Referring now back to the grammar, the grammar is a syntactic grammar such as but not limited to a context-free grammar, a N-grammar or a hybrid grammar. (Of course, DTMF grammars, handwriting grammars, gesture grammars and image grammars would be used when corresponding forms of recognition are employed. As used herein, a "grammar" includes information for performing recognition, and in a

further embodiment, information corresponding to expected input to be entered, for example, in a specific field) A new control 290 (herein identified as "reco"), comprising a first extension of the mark-up language, includes various elements, two of which are illustrated, namely a grammar element "grammar" and a "bind" element. Generally, like the code downloaded to a client from web server 202, the grammars can originate at web server 202 and be downloaded to the client and/or forwarded to a remote server for speech processing. The grammars can then be stored locally thereon in a cache. Eventually, the grammars are provided to the recognition server 204 for use in recognition. The grammar element is used to specify grammars, either inline or referenced using an attribute.

Upon receipt of recognition results from recognition server 204 corresponding to the recognized speech, handwriting, gesture, image, etc., syntax of reco control 290 is provided to receive the corresponding results and associate it with the corresponding field, which can include rendering of the text therein on display 34. In the illustrated embodiment, upon completion of speech recognition with the result sent back to the client, it deactivates the reco object and associates the recognized text with the corresponding field. Portions 282 and 284 operate similarly wherein unique reco objects and grammars are called for each of the fields 252 and 254 and upon receipt of the recognized text is associated with each of the fields 252 and 254. With respect to receipt of the card number field 252, the function "handle" checks the length of the card number with respect to the card type in a manner similar to that described above with respect to FIG. 7.

Generally, use of speech recognition in conjunction with architecture 200 and the client side mark-up language occurs as follows: first, the field that is associated with the speech to be given is indicated. In the illustrated embodiment, the stylus 33 is used; however, it should be understood that the present invention is not limited to use of the stylus 33 wherein any form of indication can be used such as buttons, a mouse pointer, rotatable wheels or the like. Corresponding event such as "onClick" can be provided as is well known with use of visual mark-up languages. It should be understood that the present invention is not limited to the use of the "onClick" event to indicate the start of voice, handwriting, gesture, etc commands. Any available GUI event can be used for the same purpose as well, such as "onSelect'. In one embodiment, such eventing is particularly useful for it serves to indicate both the beginning and/or end of the corresponding speech. It should also be noted that the field for which the speech is directed at can be indicated by the user as well as programs running on the browser that keep track of user interactions.

At this point, it should be stated that different scenarios of speech recognition require different behaviors and/or outputs from recognition server 204. Although the starting of the recognition process is standard in all cases - an explicit start () call from uplevel browsers, or a declarative <reco> element in downlevel browsers - the means for stopping speech recognition may differ.

In the example above, a user in a multimodal application will control input into the device by, for example, tapping and holding on a pressure sensitive

display. The browser then uses a GUI event, e.g. "pen-up", to control when recognition should stop and then returns the corresponding results. However, in a voice-only scenario such as in a telephone application (discussed below) or in a hands-free application, the user has no direct control over the browser, and the recognition server 204 or the client 30, must take the responsibility of deciding when to stop recognition and return the results (typically once a path through the grammar has been recognized). Further, dictation and other scenarios where intermediate results need to be returned before recognition is stopped (also known as "open microphone") not only requires an explicit stop function, but also needs to return multiple recognition results to the client 30 and/or web server 202 before the recognition process is stopped.

In one embodiment, the Reco element can include a "mode" attribute to distinguish the following three modes of recognition, which instruct the recognition server 204 how and when to return results. The return of results implies providing the "onReco" event or activating the "bind" elements as appropriate. In one embodiment, if the mode is unspecified, the default recognition mode can be "automatic".

FIG. 12 is a pictorial representation of operation of the "automatic" mode for speech recognition (similar modes, events, etc. can be provided for other forms of recognition). A timeline 281 indicates when the recognition server 204 is directed to begin recognition at 283, and where the recognition server 204 detects speech at 285 and determines that speech has ended at 287.

Various attributes of the Reco element control behavior of the recognition server 204. The attribute

"initialTimeout" 289 is the time between the start of recognition 283 and the detection of speech 285. If this time period is exceeded, "onSilence" event 291 will be provided from the recognition server 204, signaling that recognition has stopped. If the recognition server 204 finds the utterance to be unrecognizable, an "onNoReco" event 293 will be issued, which will also indicate that recognition has stopped.

Other attributes that can stop or cancel recognition include a "babbleTimeout" attribute 295, which is the period of time in which the recognition server 204 must return a result after detection of speech at 285. If exceeded, different events are issued according to whether an error has occurred or not. If the recognition server 204 is still processing audio, for example, in the case of an exceptionally long utterance, the "onNoReco" attribute 293 is issued. However, if the "babbleTimeout" attribute 295 is exceeded for any other reason, a recognizer error is more likely and an "onTimeout" event 297 is issued. Likewise, a "maxTimeout" attribute 299 can also be provided and is for the period of time between the start of recognition 283 and the results returned to the client 30. If this time period is exceeded, the "onTimeout" event 297 is issued.

If, however, a time period greater than an "endSilence" attribute 301 is exceeded, implying that recognition is complete, the recognition server 204 automatically stops recognition and returns its results. It should be noted that the recognition server 204 implement a confidence measure to determine if the recognition results should be returned. If the confidence measure is below a threshold, the "onNoReco" attribute 293 is issued, whereas if the confidence measure is above the

threshold a "onNoReco" attribute 303 and the results of recognition are issued. FIG. 12 thereby illustrates that in "automatic mode" no explicit stop () calls are made.

FIG. 13 pictorially illustrates "single mode" operation of the recognition server 204. Attributes and events described above with respect to the "automatic mode" are applicable and are so indicated with the same reference numbers. However, in this mode of operation, a stop () call 305 is indicated on timeline 281. The stop () call 305 would correspond to an event such as "pen-up" by the user. In this mode of operation, the return of a recognition result is under the control of the explicit stop () call 305. As with all modes of operation, the "onSilence" event 291 is issued if speech is not detected within the "initialTimeout" period 289, but for this mode of operation recognition is not stopped. Similarly, a "onNoReco" event 293 generated by an unrecognizable utterance before the stop () call 305 does not stop recognition. However, if the time periods associated with the "babbleTimeout" attribute 295 or the "maxTimeout" attribute 299 are exceeded recognition will stop.

FIG. 14 pictorially illustrates "multiple mode" operation of the recognition server 204. As indicated above, this mode of operation is used for an "open-microphone" or in a dictation scenario. Generally, in this mode of operation, recognition results are returned at intervals until an explicit stop () call 305 is received or the time periods associated with the "babbleTimeout" attribute 295 or the "maxTimeout" attribute 299 are exceeded. It should be noted, however, that after any "onSilence" event 291, "onReco" event 303, or "onNoReco"

event 293, which does not stop recognition, timers for the "babbleTimeout" and "maxTimeout" periods will be reset.

Generally, in this mode of operation, for each phrase that is recognized, a "onReco" event 303 is issued and the result is returned until the stop () call 305 is received. If the "onSilence" event 291 is issued due to an unrecognizable utterance these events are reported but recognition will continue.

As indicated above, the associated reco object or objects for the field is activated, which includes providing at least an indication to the recognition server 204 of which grammar to use. This information can accompany the speech data recorded at the client 30 and sent to the recognition server 204. As indicated above, speech data can comprise streaming data associated with the speech entered by the user, or can include pre-processed speech data indicating speech features that are used during speech recognition. In a further embodiment, client side processing can also include normalization of the speech data such that the speech data received by the recognition server 204 is relatively consistent from client to client. This simplifies speech processing of the recognition server 204 thereby allowing easier scalability of the recognition server 204 since the recognition server can be made stateless with respect to the type of client and communication channel.

Upon receipt of the recognition result from the recognition server 204, the recognition result is associated with the corresponding field, and client-side verification or checking can be performed, if desired. Upon completion of all of the fields associated with the code currently rendered by the client, the information is sent

to web server 202 for application processing. From the foregoing, it should be clear that although the web server 202 has provided code or pages/scripts suitable for recognition to the client 30, the recognition services are not performed by the web server 202, but rather by the recognition server 204. The invention, however, does not preclude an implementation where the recognition server 204 is collocated with the web server 202, or the recognition server 204 is part of the client 30. In other words, the extensions provided herein are beneficial even when the recognition server 204 is combined with the web server 202 or client 30 because the extension provide a simple and convenient interface between these components.

While not shown in the embodiment illustrated in FIG. 8, the reco control can also include a remote audio object (RAO) to direct the appropriate speech data to the recognition server 204. The benefit for making RAO a plug-in object is to allow a different one for each different device or client because the sound interface may likely be different. In addition, the remote audio object can allow multiple reco elements to be activated at the same time.

FIGS. 9A and 9B illustrate a voice-only mark-up language embodied herein as HTML with pages/scripts. As clearly illustrated, the code also includes a body portion 300 and a script portion 302. There is another extension of the markup language - prompt control 303 which include attributes like bargein. However, speech recognition is conducted differently in the voice-only embodiment of FIGS. 9A and 9B. The process is now controlled entirely by the script function "checkFilled" which will determine the unfilled fields and activate correspondent prompt and new objects. Nevertheless, grammars are activated using the

same context as that described above with respect to FIG. 8, wherein speech data and the indication of the grammar to use are provided to the recognition server 204. Likewise, the output received from the recognition server 204 is associated with fields of the client (herein telephony voice browser 212).

Other features generally unique to voice-only applications is an indication to the user when speech has not been recognized. In multimodal applications such as Fig 8, 'onNoReco' simply puts null value on the displayed field to indicate no-recognition, thus no further action is required. In the voice-only embodiment, "onNoReco" 305 calls or executes a function "mumble", which forwards a word phrase to recognition server 204, that in turn, is converted to speech using a suitable text-to-speech system 307 (FIG. 5). Recognition server 204 returns an audio stream to the telephony voice browser 212, which in turn, is transmitted to phone 80 to be heard by the user. Likewise, other waveform prompts embodied in the voice-only application are also converted, when necessary, to an audio stream by recognition server 204.

It should be noted that in this example after playing the welcome prompt via function "welcome", function "checkFilled" prompts the user for each of the fields and activates the appropriate grammars, including repeating the fields that have been entered and confirming that the information is correct, which includes activation of a "confirmation" grammar. Note in this embodiment, each of the reco controls is initiated from the script portion 302, rather than the body portion of the previous example.

As another aspect of the present invention, the markup language executable on different types of client

devices (e.g. multimodal and non-display, voice input based client devices such as a telephone) unifies at least one of speech-related events, GUI events and telephony events for a web server interacting with each of the client devices. This is particular advantageous for it allows significant portions of the web server application to be written generically or independent of the type of client device. An example is illustrated in FIGS. 8 and 9A, 9B with the "handle" functions.

Although not shown in Fig 9, there are two more extensions to the markup language to support telephony functionality - DTMF (Dual Tone Modulated Frequency) control and call control elements or objects. DTMF works similarly to reco control. It specifies a simple grammar mapping from keypad string to text input. For example, "1" means grocery department, "2" mean pharmacy department, etc. On the other hand, call object deals with telephony functions, like call transfer and 3rd party call. The attributes, properties, methods and events are discussed in detail in the Appendix.

FIGS. 10A and 10B illustrate yet another example of a mark-up language suitable for a voice-only mode of operation. In this embodiment, the user is allowed to have some control over when information is entered or spoken. In other words, although the system may initiate or otherwise direct the user to begin speaking, the user may offer more information than what was initially asked for. This is an example of "mixed initiative". Generally, in this form of dialog interaction, the user is permitted to share the dialog initiative with the system. Besides the example indicated above and discussed below in detail where the user provides more information then requested by a prompt,

the user could also switch tasks when not prompted to do so.

In the example of FIGS 10A and 10B, a grammar identified as "do_field" includes the information associated with the grammars "g_card_types", "g_card_num" and "g_expiry_date". In this example, telephony voice browser 212 sends speech data received from phone 80 and an indication to use the "do_field" grammar to recognition server 204 upon receipt of the recognized speech as denoted by "onReco", the function "handle" is called or executed that includes assocciating the values for any or all of the fields recognized from the speech data. In other words, the result obtained from the recognition server 204 also includes indications for each of the fields. This information is parsed and associated with the corresponding fields according to binding rules specified in 405. As indicated in FIG. 5, the recognition server 204 can include a parser 309.

From FIGS. 7, 8, 9A, 9B, 10A and 10B, a very similar web development framework is used. Data presentation is also very similar in each of these cases. In addition, the separation of data presentation and flow controls allow maximum reusability between different applications (system initiative and mixed-initiative), or different modalities (GUI web-based, voice-only and multimodal). This also allows a natural extension from voice-only operation through a telephone to a multimodal operation when phones include displays and functionalities similar to device 30. Appendix A provides further details of the controls and objects discussed above.

As indicated above, uplevel browsers can use scripting in order to perform various needs such as

invoking the function "handle" in the above examples to assign the recognition results. In the embodiments described above and as further described in Appendix A at 2.1.2, the "bind" element will parse the recognition results and assign values wherein the "bind" element is a subelement or child element of the "reco" element.

Although scripting can be useful, it is believed by many not to be always the best form of browser implementation due, for example, to security concerns. Therefore, in yet another embodiment or aspect of the present invention, the "bind" element is a high level element (similar to "reco") and is provided with other richer properties, which can in effect mimic scripting without scripting per se.

Without using scripting or without using the aspect of the present invention discussed below, some of the capabilities discussed below such as sophisticated dialog effects could only be accomplished by submitting a page back to the web server 202, executing the application logic thereon to generate a new page, and sending the page back to the client device. This aspect of the present invention allows a programmer to invoke methods on objects of the page without incurring a server roundtrip.

In the embodiments described above, the "bind" element has only the attributes "TargetElement" and "TargetAttribute" for assigning recognition results to a field in the form or web page. In a further embodiment, the "bind" element also includes a "TargetMethod", which is added for object method invocation. The use and capability of "TargetMethod" is the principal technique for mimicking scripting. For example, the following syntax can be used to invoke the "X" method of the object "OBJ1":

<bind TargetElement = "OBJ1" TargetMethod = "X"...>.

Note that although the examples shown here follow the HTML/XHTML event syntax, those skilled at art should find it straightforward to generalize the <bind> usage to other eventing mechanisms, including but not limited to W3C Document Object Model Level 2 or Level 3 eventing standard, ECMA Common language Infrastructure (CLI) event model, Java programming language event model, W3C Synchronous Multimedia Integration Language (SMIL), and the upcoming W3C XML Events standard proposal.

FIGS. 15A and 15B are a page of mark-up language executable on a client, particularly a downlevel browser. In this example, the user is asked through audio prompts for a drink. The system then confirms what drink was ordered. Depending on the recognition results, the "bind" element guides the execution using declared logic. When the drink is confirmed, the form is submitted back to the web server 202, all without scripting.

Generally, the mark-up example of FIGS. 15A and 15B includes a data section 350, a speech section 352 and user interface sections 354, 356 and 358. Section 354 receives the recognition result from the general inquiry as to what drink the user would like and directs interactive recognition flow to either reprompting, inquiring as to whether cream or sugar is needed, or confirming the drink ordered. In particular, section 356 receives a recognition result when cream or sugar is also ordered. Section 358 receives the recognition result for confirmation of the drink. Section 360 is a call control section employing a new messaging object "SMEX", which is discussed further below.

As indicated above, the "bind" element of this aspect of the present invention includes object method invocation, which will initiate user interaction in the example of FIGS. 15A and 15B by playing the "welcome" prompt when the "start" method on the "welcome" object is executed at 361.

The user is then asked "Do you want coke, coffee or orange juice?" by executing the "start" method of the "asked" object at 362. Recognition is then performed by invoking the "start" method on the recognition "reco_drink" object at 363.

The mark-up of section 354 is then executed wherein the grammar used by the recognition server 204 is provided by the Xpath statement "./drink types". Note that although this example utilizes W3C Xpath language, those skilled at art should find it straightforward to extend the concept to other standard languages, including but not limited to W3C XML Query Language (XQL). As specified by the "bind" element 364, if the recognition result received from the recognition server 204 has a confidence score less than 10, the prompt object "reprompt" is executed at 366, followed by the prompt object "ask" 368, at which point, the recognition object "reco_drink" is reinitiated at 370. If the recognition result returned is "coffee" having a confidence greater than 10, the field "drink" is assigned the value of the recognition result at 372 and the user is then prompted as to whether or not he/she would like cream or sugar by the prompt object "cream_sugar" at 374. The recognition object "reco_cream sugar" in section 356 is then invoked at 376. Otherwise, if the confidence score of the recognition result is greater than 10, but not coffee, the field drink is then again assigned at 378. Confirmation

of the recognition result is provided at 380 by executing the prompt object "confirm", followed by invocation of the recognition object "reco_yesno" in section 358 at 382. If the user answers "yes" with a confidence score greater than 10, the prompt object "thanks" is played at 384 and then the form is then submitted at 386. Otherwise, if the user answers "no" or the confidence score of the recognition result is less than 10, the prompt object "retry" is executed at 390, once again followed by the prompt object "ask" being executed at 392, and invocation of the "reco_drink" recognition object at 394.

From the foregoing example, the "bind" element allows multiple invocations of methods as indicated in sections 354, 356 or 358. If desired, multiple assignments of the recognized result can also be declared. In the embodiment illustrated, when multiple assignments and method invocations are declared, they are executed in the document order.

In a further embodiment, a convention for passing method arguments is also provided. In other words, some methods may require a list of arguments. This is achieved by using "arg" subelement. For example, given the following mark-up:

<bind TargetElement = "OBJ" TargetMethod = "F"><arg>X</arg><arg>Y</arg></bind>

is equivalent to "OBJ.F(X,Y)", or "OBJ" is an object that has a method "F" with parameters or arguments "X" and "Y".

The "bind" element can also include an "event" attribute, which declares the bind element is intended for which event. For example, the mark-up:

<bind event = "onNoReco" = TargetElement = "prompt1" TargetMethod = "start"/>

means when the "onNoReco" event is sent, the method "start" of the object "prompt1" will be invoked. To be consistent with use of the "bind" element as a child element of the "Reco" element as described above for example with respect to FIG. 8, the default attribute for the "bind" element is "onReco".

The "bind" element as a high level element can include any of the events specified in section 2.4 of the Appendix. In addition, the "bind" element can also include a "onError" event having a "status" attribute that can be accessed and used to direct program flow. To the extent other events of the "bind" element have "status" attributes, these can be accessed as well.

In addition to checking the conditions of the recognition result, the current document or page being executed can also be checked as well. In particular, both the "test" and the "value" attribute can be extended to include a "host" primitive that refers to the root node of the containing document. For example, referring back to FIGS. 15A, 15D, the example contained therein has additional logic at section 354 to ask whether the user wants cream or sugar when he/she asks for coffee. The flags for adding cream or sugar and thus invocation of section 356, will be turned on only if the drink field is "coffee" as specified by the markup "host()/get_drink/drink='coffee'".

It should also be noted, the "bind" element is not only applicable to recognition results from the speech server 204 and receive or assign values in the document, but also to message objects (herein denoted by "smex", for example, from applications running on the client device. In the example of FIGS. 15A and 15B, the page is executed when

a telephony application running on the client device detects a call. In section 360, the "bind" element executes or plays the "welcome" prompt and begins recognition by executing the "reco_drink" object, when the message "/Call_connected" is received. Like the recognition results received from the speech server 204, the messages received can vary greatly. Some of the messages are well-defined in order to initiate desired program flow. Others can be received and processed (e.g. parsed just like received recognition results of the recognition server. For example, this allows the markup to be used like a natural language parser of text from a keyboard. The reco element in Appendix A includes a property for performing this function. Likewise, the prompt element can be used to provide a textual message for dynamic content or audio wave files by using the property "innertext", also further explained in the Appendix A. Eventing can be similar to eventing for recognition results. For example, eventing can include "onReceived", which gets sent when the message source (e.g. application running on the client device) has a message available for the browser.

The "smex" or message object thus allows the markup tags discussed herein to be extended to other components or applications running on the client device. As another example, the message object can be used to communicate to a TTY component for the hearing impaired running on the client device. Rather than using speech recognition, the TTY component will provide a message of what the user has typed. This message is then used as if a recognition result was received from the recognition server wherein the message can be parsed and assigned to fields of the form, or other processing can take place using the

"reco", "grammar" or "bind" elements described above. Further discussion of the message or "smex" object is provided in Appendix A.

The "bind" element can also include a "for" attribute, which allows its action to be attached to other objects on the page. For example, a mark-up such as:

<bind for = "prompt 1" event = "onComplete" targetElement = "prompt 2" = targetMethod = "start" />.

will invoke the start method of the object "prompt 2" when the object "prompt 1" sends the event "onComplete".

Referring back to FIG. 5, web server 202 can include a server side plug-in declarative authoring tool or module 320 (e.g. ASP or ASP+ by Microsoft Corporation, JSP, or the like). Server side plug-in module 320 can dynamically generate client-side mark-ups and even a specific form of mark-up for the type of client accessing the web server 202. The client information can be provided to the web server 202 upon initial establishment of the client/server relationship, or the web server 202 can include modules or routines to detect the capabilities of the client. In this manner, server side plug-in module 320 can generate a client side mark-up for each of the voice recognition scenarios, i.e. voice only through phone 80 or multimodal for device 30. By using a consistent client side model (reco and prompt controls that can be used in each application), application authoring for many different clients is significantly easier.

In addition to dynamically generating client side mark-ups, high-level dialog modules, like getting credit card information illustrated in FIG. 6 with a mark-up examples of FIGS. 8, 9A and 9B, can be implemented as a

server-side control as stored in store 324 for use by developers in application authoring. In general, the high-level dialog modules 324 would generate dynamically client-side markup and script in both voice-only and multimodal scenarios based on parameters specified by developers. The high-level dialog modules can include parameters to generate client-side mark-ups to fit the developers' needs. For example, a credit card information module can include a parameter indicating what types of credit cards the client/side mark-up script should allow. A sample ASP+ page using in server side plug-in module 320 is illustrated in FIG. 11.

Although the present invention has been described with reference to preferred embodiments, workers skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.

APPENDIX A

1 Introduction

The following tags are a set of markup elements that allows a document to use speech as an input or output medium. The tags are designed to be self-contained XML that can be imbedded into any SGML derived markup languages such as HTML, XHTML, cHTML, SMIL, WML and the like. The tags used herein are similar to SAPI 5.0, which are known methods available from Microsoft Corporation of Redmond, Washington. The tags, elements, events, attributes, properties, return values, etc. are merely exemplary and should not be considered limiting. Although exemplified herein for speech and DTMF recognition, similar tags can be provided for other forms of recognition.

The main elements herein discussed are:

<prompt ...> for speech synthesis configuration and prompt playing
<reco ...> for recognizer configuration and recognition execution and post-processing
<grammar ...> for specifying input grammar resources
<bind ...> for processing of recognition results
<dtmf ...> for configuration and control of DTMF

2 Reco

The Reco element is used to specify possible user inputs and a means for dealing with the input results. As such, its main elements can be <grammar> and <bind>, and it contains resources for configuring recognizer properties.

Reco elements are activated programmatically in uplevel browsers via Start and Stop methods, or in SMIL-enabled browsers by using SMIL commands. They are considered active declaratively in downlevel browsers (i.e. non script-supporting browsers) by their presence on the page. In order to permit the activation of multiple grammars in parallel, multiple Reco elements may be considered active simultaneously.

Recos may also take a particular mode - 'automatic', 'single' or 'multiple' - to distinguish the kind of recognition scenarios which they enable and the behavior of the recognition platform.

2.1 Reco content

The Reco element contains one or more grammars and optionally a set of bind elements which inspect the results of recognition and copy the relevant portions to values in the containing page.

In uplevel browsers, Reco supports the programmatic activation and deactivation of individual grammar rules. Note also that all top-level rules in a grammar are active by default for a recognition context.

2.1.1 <grammar> element

The grammar element is used to specify grammars, either inline or referenced using the src attribute. At least one grammar (either inline or referenced) is typically specified. Inline grammars can be text-based grammar formats, while referenced grammars can be text-based or binary type. Multiple grammar elements may be specified. If

more than one grammar element is specified, the rules within grammars are added as extra rules within the same grammar. Any rules with the same name will be overwritten.

Attributes:

- **src**: Optional if inline grammar is specified. URI of the grammar to be included. Note that all top-level rules in a grammar are active by default for a recognition context.
- **langID**: Optional. String indicating which language speech engine should use. The string format follows the xml:lang definition. For example, langID="en-us" denotes US English. This attribute is only effective when the langID is not specified in the grammar URI. If unspecified, defaults to US English.

  If the langID is specified in multiple places then langID follows a precedence order from the lowest scope - remote grammar file (i.e language id is specified within the grammar file) followed by grammar element followed by reco element.

```
<grammar src="FromCity.xml" />
    or
<grammar>
    <rule toplevel="active">
        <p>from </p>
        <ruleref name="cities" />
    </rule>
    <rule name="cities">
        <l>
        <p> Cambridge </p>
        <p> Seattle </p>
        <p> London </p>
        </l>
    </rule>
```

```
</grammar>
```

If both a src-referenced grammar and an inline grammar are specified, the inline rules are added to the referenced rules, and any rules with the same name will be overwritten.

2.1.2 <bind> element

The bind element is used to bind values from the recognition results into the page.

The recognition results consumed by the bind element can be an XML document containing a semantic markup language (SML) for specifying recognition results. Its contents include semantic values, actual words spoken, and confidence scores. SML could also include alternate recognition choices (as in an N-best recognition result). A sample SML document for the utterance "I'd like to travel from Seattle to Boston" is illustrated below:

```
<sml confidence="40">
    <travel text="I'd like to travel from Seattle to
Boston">
        <origin_city confidence="45"> Seattle
</origin_city>
        <dest_city   confidence="35"> Boston
</dest_city>
    </travel>
</sml>
```

Since an in-grammar recognition is assumed to produce an XML document - in semantic markup language, or SML - the values to be bound from the SML document are referenced using an XPath query. And since the elements in the page into which the values will be bound should be are uniquely identified

(they are likely to be form controls), these target elements are referenced directly.

Attributes:

- **targetElement**: Required. The element to which the *value* content from the SML will be assigned (as in W3C SMIL 2.0).
- **targetAttribute**: Optional. The attribute of the target element to which the *value* content from the SML will be assigned (as with the *attributeName* attribute in SMIL 2.0). If unspecified, defaults to "value".
- **test**: Optional. An ***XML Pattern*** (as in the W3C XML DOM specification) string indicating the condition under which the recognition result will be assigned. Default condition is true.
- **value**: Required. An ***XPATH*** (as in the W3C XML DOM specification) string that specifies the value from the recognition result document to be assigned to the target element.

Example:
So given the above SML return, the following reco element uses bind to transfer the values in origin_city and dest_city into the target page elements txtBoxOrigin and txtBoxDest:

```
<input name="txtBoxOrigin" type="text"/>
<input name="txtBoxDest" type="text" />

<reco id="travel">
    <grammar src="./city.xml" />
```

```
    <bind      targetElement="txtBoxOrigin"
          value="//origin_city" />
    <bind      targetElement="txtBoxDest"
          value="//dest_city" />
</reco>
```

This binding may be conditional, as in the following example, where a test is made on the confidence attribute of the dest_city result as a pre-condition to the bind operation:

```
<bind targetElement="txtBoxDest"
     value="//dest_city"
     test="/sml/dest_city[@confidence $gt$ 40]"
/>
```

The bind element is a simple declarative means of processing recognition results on downlevel or uplevel browsers. For more complex processing, the reco DOM object supported by uplevel browsers implements the onReco event handler to permit programmatic script analysis and post-processing of the recognition return.

2.2 Attributes and properties

The following attributes are supported by all browsers, and the properties by uplevel browsers.

2.2.1 Attributes

The following attributes of Reco are used to configure the speech recognizer for a dialog turn.

- **initialTimeout**: Optional. The time in milliseconds between start of recognition and the detection of speech. This value is passed to the recognition platform, and if exceeded, an onSilence event will be provided from the recognition platform (see 2.4.2). If not specified, the speech platform will use a default value.
- **babbleTimeout**: Optional. The period of time in milliseconds in which the recognizer must return a result after detection of speech. For recos in automatic and single mode, this applies to the period between speech detection and the stop call. For recos in 'multiple' mode, this timeout applies to the period between speech detection and each recognition return - i.e. the period is restarted after each return of results or other event. If exceeded, different events are thrown according to whether an error has occurred or not. If the recognizer is still processing audio – eg in the case of an exceptionally long utterance - the onNoReco event is thrown, with status code 13 (see 2.4.4). If the timeout is exceeded for any other reason, however, a recognizer error is more likely, and the onTimeout event is thrown. If not specified, the speech platform will default to an internal value.
- **maxTimeout**: Optional. The period of time in milliseconds between recognition start and results returned to the browser. If exceeded, the onTimeout event is thrown by the browser - this caters for network or recognizer failure in distributed

environments. For recos in 'multiple' mode, as with babbleTimeout, the period is restarted after the return of each recognition or other event. Note that the maxTimeout attribute should be greater than or equal to the sum of initialTimeout and babbleTimeout. If not specified, the value will be a browser default.

- **endSilence:** Optional. For Recos in automatic mode, the period of silence in milliseconds after the end of an utterance which must be free of speech after which the recognition results are returned. Ignored for recos of modes other than automatic. If unspecified, defaults to platform internal value.
- **reject:** Optional. The recognition rejection threshold, below which the platform will throw the 'no reco' event. If not specified, the speech platform will use a default value. Confidence scores range between 0 and 100 (integer). Reject values lie in between.
- **server:** Optional. URI of speech platform (for use when the tag interpreter and recognition platform are not co-located). An example value might be *server=protocol://yourspeechplatform*. An application writer is also able to provide speech platform specific settings by adding a querystring to the URI string, eg *protocol://yourspeechplatform?bargeinEnergyThreshold=0.5*.
- **langID:** Optional. String indicating which language speech engine should use. The string format follows the xml:lang definition. For example, langID="en-us" denotes US English. This attribute is only effective when the langID is not specified in the grammar element (see 2.1.1).
- **mode:** Optional. String specifying the recognition mode to be followed. If unspecified, defaults to "automatic" mode.

2.2.2 Properties

The following properties contain the results returned by the recognition process (these are supported by uplevel browsers).

- **recoResult** Read-only. The results of recognition, held in an XML DOM node object containing semantic markup language (SML), as described in 2.1.2, In case of no recognition, the property returns null.
- **text** Read/Write. A string holding the text of the words recognized (i.e., a shorthand for contents of the text attribute of the highest level element in the SML recognition return in recoResult in a read mode. In write mode, a string can be assigned, which will then be parsed as if the string corresponds to the recognition result. The write mode allows extension of the markup tags and processing thereof to other components or applications on the client device. The string can be obtained from the "smex" message object.
- **status:** Read-only. Status code returned by the recognition platform. Possible values are 0 for successful recognition, or the failure values -1 to -4 (as defined in the exceptions possible on the Start method (section 2.3.1) and Activate method (section

2.3.4)), and statuses -11 to -15 set on the reception of recognizer events (see 2.4).

2.3 Object methods

Reco activation and grammar activation may be controlled using the following methods in the Reco's DOM object. With these methods, uplevel browsers can start and stop Reco objects, cancel recognitions in progress, and activate and deactivate individual grammar top-level rules (uplevel browsers only).

2.3.1 Start

The Start method starts the recognition process, using as active grammars all the top-level rules for the recognition context which have not been explicitly deactivated.

Syntax:

Object.Start( )

Return value:

None.

Exception:

The method sets a non-zero status code and fires an onNoReco event when fails. Possible failures include no grammar (reco status = -1), failure to load a grammar, which could be a variety of reasons like failure to compile grammar, non-existent URI (reco status = -2), or speech platform errors (reco status = -3).

2.3.2 Stop

The Stop method is a call to end the recognition process. The Reco object stops recording audio, and the recognizer

returns recognition results on the audio received up to the point where recording was stopped. All the recognition resources used by Reco are released, and its grammars deactivated. (Note that this method need not be used explicitly for typical recognitions in automatic mode, since the recognizer itself will stop the reco object on endpoint detection after recognizing a complete sentence.) If the Reco has not been started, the call has no effect.

Syntax:

Object.Stop( )

Return value:

None.

Exception:

None.

2.3.3 Cancel

The Cancel method stops the audio feed to the recognizer, deactivates the grammar and releases the recognizer and discards any recognition results. The browser will disregard a recognition result for canceled recognition. If the recognizer has not been started, the call has no effect.

Syntax:

Object.Cancel( )

Return value:

None.

Exception:

None.

2.3.4 Activate

The Activate method activates a top-level rule in the context free grammar (CFG). Activation must be called before recognition begins, since it will have no effect during a 'Started' recognition process. Note that all the grammar top-level rules for the recognition context which have not been explicitly deactivated are already treated as active.

Syntax:

Object.Activate(strName);

Parameters:

- **strName:** Required. Rule name to be activated.

Return value:

None.

Exception:

None.

2.3.5 Deactivate

The method deactivates a top-level rule in the grammar. If the rule does not exist, the method has no effect.

Syntax:

Object.Deactivate(strName);

Parameters:

- strName: Required. Rule name to be deactivated. An empty string deactivates all rules.

Return value

None.

Exception

None.

2.4 Reco events

The Reco DOM object supports the following events, whose handlers may be specified as attributes of the reco element.

2.4.1 onReco:

This event gets fired when the recognizer has a recognition result available for the browser. For recos in automatic mode, this event stops the recognition process automatically and clears resources (see 2.3.2). OnReco is typically used for programmatic analysis of the recognition result and processing of the result into the page.

Syntax:

Inline HTML	<Reco onReco ="*handler*" >
Event property	Object.onReco = *handler*; Object.onReco = GetRef("*handler*");

Event Object Info:

Bubbles	No
To invoke	User says something
Default action	Return recognition result object

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data (see the use of the event object in the example below).

Example

The following XHTML fragment uses onReco to call a script to parse the recognition outcome and assign the values to the proper fields.

```
<input name="txtBoxOrigin" type="text" />
<input name="txtBoxDest" type="text" />
<reco onReco="processCityRecognition()"/>
    <grammar src="/grammars/cities.xml" />
</reco>

<script><![CDATA[
    function processCityRecognition () {
        smlResult = event.srcElement.recoResult;

        origNode =
smlResult.selectSingleNode("//origin_city");
        if (origNode != null) txtBoxOrigin.value =
origNode.text;

        destNode =
smlResult.selectSingleNode("//dest_city");
        if (destNode != null) txtBoxDest.value =
destNode.text;
    }
]]></script>
```

2.4.2 onSilence:

onSilence handles the event of no speech detected by the recognition platform before the duration of time specified in the initialTimeout attribute on the Reco (see 2.2.1).

This event cancels the recognition process automatically for the automatic recognition mode.

Syntax:

Inline HTML	<reco onSilence="*handler*" …>
Event property (in ECMAScript)	Object.onSilence = *handler* Object.onSilence = GetRef("*handler*");

Event Object Info:

Bubbles	No
To invoke	Recognizer did not detect speech within the period specified in the initialTimeout attribute.
Default action	Set status = -1

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

2.4.3 onTimeout

onTimeout handles two types of event which typically reflect errors from the speech platform.

- It handles the event thrown by the tags interpreter which signals that the period specified in the maxtime attribute (see 2.2.1) expired before recognition was completed. This event will typically reflect problems that could occur in a distributed architecture.

- It also handles (ii) the event thrown by the speech recognition platform when recognition has begun but processing has stopped without a recognition within the period specified by babbleTimeout (see 2.2.1).

This event cancels the recognition process automatically.

Syntax:

Inline HTML	<reco onTimeout="*handler*" ..>
Event property (in ECMAScript)	Object.onTimeOut = *handler* Object.onTimeOut = GetRef("*handler*");

Event Object Info:

Bubbles	No
To invoke	Thrown by the browser when the period set by the maxtime attribute expires before recognition is stopped.
Default action	Set reco status to -12.

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

2.4.4 onNoReco:

onNoReco is a handler for the event thrown by the speech recognition platform when it is unable to return valid recognition results. The different cases in which this may

happen are distinguished by status code. The event stops the recognition process automatically.

Syntax:

Inline HTML	<Reco onNoReco ="*handler*" >
Event property	Object.onNoReco = *handler*; Object.onNoReco = GetRef("*handler*");

Event Object Info:

Bubbles	No
To invoke	Recognizer detects sound but is unable to interpret the utterance.
Default action	Set status property and return null recognition result. Status codes are set as follows: **status -13**: sound was detected but no speech was able to be interpreted; **status -14**: some speech was detected and interpreted but rejected with insufficient confidence (for threshold setting, see the reject attribute in 2.2.1). **status -15**: speech was detected and interpreted, but a complete recognition was unable to be returned between the detection of speech and the duration specified in the babbleTimeout attribute (see 2.2.1).

Event Properties:
Although the event handler does not receive properties directly, the handler can query the event object for data.

3 Prompt

The prompt element is used to specify system output. Its content may be one or more of the following:

- inline or referenced text, which may be marked up with prosodic or other speech output information;
- variable values retrieved at render time from the containing document;
- links to audio files.

Prompt elements may be interpreted declaratively by downlevel browsers (or activated by SMIL commands), or by object methods on uplevel browsers.

3.1 Prompt content

The prompt element contains the resources for system output, either as text or references to audio files, or both.

Simple prompts need specify only the text required for output, eg:

```
<prompt id="Welcome">
   Thank you for calling ACME weather report.
</prompt>
```


This simple text may also contain further markup of any of the kinds described below.

3.1.1 Speech Synthesis markup

Any format of speech synthesis markup language can be used inside the prompt element. (This format may be specified in the 'tts' attribute described in 3.2.1.) The following example shows text with an instruction to emphasize certain words within it:

```
<prompt id="giveBalance">
     You have <emph> five dollars </emph> left in your
account.
</prompt>
```

3.1.2 Dynamic content

The actual content of the prompt may need to be computed on the client just before the prompt is output. In order to confirm a particular value, for example, the value needs to be dereferenced in a variable. The value element may be used for this purpose.

Value Element

value: Optional. Retrieves the values of an element in the document.

Attributes:

- **targetElement**: Optional. Either href or targetElement must be specified. The id of the element containing the value to be retrieved.
- **targetAttribute**: Optional. The attribute of the element from which the value will be retrieved.

- **href**: Optional. The URI of an audio segment. href will override targetElement if both are present.

The targetElement attribute is used to reference an element within the containing document. The content of the element whose id is specified by targetElement is inserted into the text to be synthesized. If the desired content is held in an attribute of the element, the targetAttribute attribute may be used to specify the necessary attribute on the targetElement. This is useful for dereferencing the values in HTML form controls, for example. In the following illustration, the "value" attributes of the "txtBoxOrigin" and "txtBoxDest" elements are inserted into the text before the prompt is output

```
<prompt id="Confirm">
    Do you want to travel from
    <value targetElement="txtBoxOrigin"
targetAttribute="value" />
    to
    <value targetElement="txtBoxDest"
targetAttribute="value" />
    ?
</prompt>
```

3.1.3 Audio files

The value element may also be used to refer to a pre-recorded audio file for playing instead of, or within, a synthesized prompt. The following example plays a beep at the end of the prompt:

```
<prompt>
```



```
    After the beep, please record your message.
    <value href="/wav/beep.wav" />
</prompt>
```

3.1.4 Referenced prompts

Instead of specifying content inline, the src attribute may be used with an empty element to reference external content via URI, as in:

```
<prompt id="Welcome" src="/ACMEWeatherPrompts#Welcome"
/>
```

The target of the src attribute can hold any or all of the above content specified for inline prompts.

3.2 Attributes and properties

The prompt element holds the following attributes (downlevel browsers) and properties (downlevel and uplevel browsers).

3.2.1 Attributes

- **tts**: Optional. The markup language type for text-to-speech synthesis. Default is "SAPI 5".
- **src**: Optional if an inline prompt is specified. The URI of a referenced prompt (see 3.1.4).
- **bargein**: Optional. Integer. The period of time in milliseconds from start of prompt to when playback can be interrupted by the human listener. Default is infinite, i.e., no bargein is allowed. Bargein=0 allows immediate bargein. This applies to whichever kind of barge-in is supported by platform. Either keyword or energy-based bargein times can be

configured in this way, depending on which is enabled at the time the reco is started.

- **prefetch**: Optional. A Boolean flag indicating whether the prompt should be immediately synthesized and cached at browser when the page is loaded. Default is false.

3.2.2 Properties

Uplevel browsers support the following properties in the prompt's DOM object.

- **bookmark**: Read-only. A string object recording the text of the last synthesis bookmark encountered.
- **status**: Read-only. Status code returned by the speech platform.

- **Innertext**: Read-only. This property will provide the text transcription of the prompt, which would be sent to the synthesizer. For instance, if a prompt comprises playing an audio wave file, this property provides a text version of that prompt (often stored with the audio wave file), which can then be displayed, or otherwise used, for example, by providing the text version of the prompt to a component or application running on the client device. The innertext property can also be used to provide text versions of prompts containing dynamic content.

3.3 Prompt methods

Prompt playing may be controlled using the following methods in the prompt's DOM object. In this way, uplevel browsers can start and stop prompt objects, pause and resume prompts

in progress, and change the speed and volume of the synthesized speech.

3.3.1 Start

Start playback of the prompt. Unless an argument is given, the method plays the contents of the object. Only a single prompt object is considered 'started' at a given time, so if Start is called in succession, all playbacks are played in sequence.

Syntax:

Object.Start([strText]);

Parameters:

- **strText:** the text to be sent to the synthesizer. If present, this argument overrides the contents of the object.

Return value:

None.

Exception:

Set status = -1 and fires an onComplete event if the audio buffer is already released by the server.

3.3.2 Pause

Pause playback without flushing the audio buffer. This method has no effect if playback is paused or stopped.

Syntax:

Object.Pause( );

Return value:

None.

Exception:

None.

3.3.3 Resume

Resume playback without flushing the audio buffer. This method has no effect if playback has not been paused.

Syntax:

Object.Resume( );

Return value:

None.

Exception:

Throws an exception when resume fails.

3.3.4 Stop

Stop playback, if not already, and flush the audio buffer. If the playback has already been stopped, the method simply flushes the audio buffer.

Syntax:

Object.Stop( );

Return value:

None.

Exception:

None.

3.3.5 Change

Change speed and/or volume of playback. Change may be called during playback.

Syntax:

Object.Change(speed, volume);

Parameters:

- **speed:** Required. The factor to change. Speed=2.0 means double the current rate, speed=0.5 means halve the current rate, speed=0 means to restore the default value.
- **volume:** Required. The factor to change. Volume=2.0 means double the current volume, volume =0.5 means halve the current volume, volume =0 means to restore the default value.

Return value:

None.

Exception:

None.

3.3.6 Prompt control example

The following example shows how control of the prompt using the methods above might be authored for a platform which does not support a keyword barge-in mechanism.

```
<html>
<title>Prompt control</title>
<head>
 <script>
    <!--
        function checkKWBargein() {
          news.change(1.0, 0.5); // turn down the volume
    while verifying
          if (keyword.text == "") { // result is below
    threshold
                news.change(1.0, 2.0); // restore the volume
                keyword.Start(); // restart the recognition
          } else {
                news.Stop(); // keyword detected! Stop the
    prompt
                // Do whatever that is necessary
          }
        }
 //
   </script>
 <script for="window" event="onload">
    <!--
```

```
        news.Start(); keyword.Start();
    //
  </script>
   </head>
<body>
 <prompt id="news" bargein="0">
   Stocks turned in another lackluster performance Wednesday
   as investors received little incentive to make any big moves
   ahead of next week's Federal Reserve meeting. The tech-heavy
   Nasdaq Composite Index dropped 42.51 points to close at
   2156.26. The Dow Jones Industrial Average fell 17.05 points
   to 10866.46 after an early-afternoon rally failed.
  <!--
 </prompt>
 <reco     id="keyword"
       reject="70"
       onReco="checkKWBargein()" >
       <grammar src=http://denali/news_bargein_grammar.xml />
  </reco>
</body>
</html>
```

3.4 Prompt events

The prompt DOM object supports the following events, whose handlers may be specified as attributes of the prompt element.

3.4.1 onBookmark

Fires when a synthesis bookmark is encountered. The event does not pause the playback.

Syntax:

Inline HTML	<prompt onBookmark="*handler*" …>
Event property	Object.onBookmark = *handler* Object.onBookmark = GetRef("*handler*");

Event Object Info:

Bubbles	No
To invoke	A bookmark in the rendered string is encountered
Default action	Returns the bookmark string

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

3.4.2 onBargein:

Fires when a user's barge-in event is detected. (Note that determining what constitutes a barge-in event, eg energy detection or keyword recognition, is up to the platform.) A specification of this event handler does not automatically turn the barge-in on.

Syntax:

Inline HTML	<prompt onBargein="*handler*" …>
Event property	Object.onBargein = *handler* Object.onBargein = GetRef("*handler*");

Event Object Info:

Bubbles	No
To invoke	A bargein event is encountered

Default action	None

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

3.4.3 onComplete:

Fires when the prompt playback reaches the end or exceptions (as defined above) are encountered.

Syntax:

Inline HTML	<prompt onComplete="*handler*" …>
Event property	Object. onComplete = *handler* Object. onComplete = GetRef("*handler*");

Event Object Info:

Bubbles	No
To invoke	A prompt playback completes
Default action	Set status = 0 if playback completes normally, otherwise set status as specified above.

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

3.4.4 Using bookmarks and events

The following example shows how bookmark events can be used to determine the semantics of a user response - either a correction to a departure city or the provision of a destination city - in terms of when bargein happened during the prompt output. The onBargein handler calls a script which sets a global 'mark' variable to the last bookmark encountered in the prompt, and the value of this 'mark' is used in the reco's postprocessing function ('heard') to set the correct value.

```
<script><![CDATA[
     var mark;
     function interrupt( ) {
          mark = event.srcElement.bookmark;
     }
     function ProcessCityConfirm() {
          confirm.stop(); // flush the audio buffer
          if (mark == "mark_origin_city")
               txtBoxOrigin.value =
event.srcElement.text;
          else
               txtBoxDest.value =
event.srcElement.text;
     }
]]></script>
<body>
<input name="txtBoxOrigin" value="Seattle"
type="text"/>
<input name="txtBoxDest" type="text" />
…
<prompt id="confirm" onBargein="interrupt()"
bargein="0">
     From <bookmark mark="mark_origin_city" />
     <value targetElement="orgin"
targetAttribute="value" />,
     please say <bookmark mark="mark_dest_city" /> the
     destination city you want to travel to.
</prompt>
<reco onReco="ProcessCityConfirm()" >
     <grammar src="/grm/1033/cities.xml" />
</reco>
```

```
...
</body>
```

4 DTMF

Creates a DTMF recognition object. The object can be instantiated using inline markup language syntax or in scripting. When activated, DTMF can cause prompt object to fire a barge-in event. It should be noted the tags and eventing discussed below with respect to DTMF recognition and call control discussed in Section 5 generally pertain to interaction between the voice browser 216 and media server 214.

4.1 Content

- **dtmfgrammar**: for inline grammar.
- **bind**: assign DTMF conversion result to proper field.

Attributes:

- **targetElement**: Required. The element to which a partial recognition result will be assigned to (cf. same as in W3C SMIL 2.0).
- **targetAttribute**: the attribute of the target element to which the recognition result will be assigned to (cf. same as in SMIL 2.0). Default is "value".
- **test**: condition for the assignment. Default is true.

Example 1: map keys to text

```
<input type="text" name="city"/>
```



```
<DTMF id="city_choice" timeout="2000" numDigits="1">
     <dtmfgrammar>
          <key value="1">Seattle</key>
          <key value="2">Boston</key>
     </dtmfgrammar>
     <bind targetElement="city" targetAttribute="value"
/>
</DTMF>
```

When "city_choice" is activated, "Seattle" will be assigned to the input field if the user presses 1, "Boston" if 2, nothing otherwise.

Example 2: How DTMF can be used with multiple fields.

```
<input type="text" name="area_code" />
<input type="text" name="phone_number" />
<DTMF id="areacode" numDigits="3"
onReco="extension.Activate()">
     <bind targetElement="area_code" />
</DTMF>
<DTMF id="extension" numDigits="7">
     <bind targetElement="phone_number" />
</DTMF>
```

This example demonstrates how to allow users entering into multiple fields.

Example 3: How to allow both speech and DTMF inputs and disable speech when user starts DTMF.

```
<input type="text" name="credit_card_number" />
<prompt onBookmark="dtmf.Start(); speech.Start()"
     bargein="C">
     Please say <bookmark name="starting" />
     or enter your credit card number now
</prompt>
<DTMF id="dtmf" escape="#" length="16"
interdigitTimeout="2000"
     onkeypress="speech.Stop()">
     <bind targetElement="credit_card_number" />
```

```
</DTMF>
<reco id="speech" >
     <grammar src="/grm/1033/digits.xml" />
     <bind targetElement="credit_card_number" />
</reco>
```

4.2 Attributes and properties

4.2.1 **Attributes**

- **dtmfgrammar**: Required. The URI of a DTMF grammar.

4.2.2 **Properties**

- **DTMFgrammar** Read-Write.

  An XML DOM Node object representing DTMF to string conversion matrix (also called DTMF grammar). The default grammar is

```
<dtmfgrammar>
     <key value="0">0</key>
     <key value="1">1</key>
     ...
     <key value="9">9</key>
     <key value="*">*</key>
     <key value="#">#</key>
</dtmfgrammar >
```

- **flush**

  Read-write, a Boolean flag indicating whether to automatically flush the DTMF buffer on the underlying telephony interface card before activation. Default is false to enable type-ahead.

- escape

  Read-Write. The escape key to end the DTMF reading session. Escape key is one key.

- **numDigits**

  Read-Write. Number of key strokes to end the DTMF reading session. If both escape and length are specified, the DTMF session is ended when either condition is met.

- **dtmfResult**

  Read-only string, storing the DTMF keys user has entered. Escape is included in result if typed.

- **text**

  Read-only string storing white space separated token string, where each token is converted according to DTMF grammar.

- **initialTimeout**

  Read-Write. Timeout period for receiving the first DTMF keystoke, in milliseconds. If unspecified, defaults to the telephony platform's internal setting.

- **interdigitTimeout**

  Read-Write. Timeout period for adjacent DTMF keystokes, in milliseconds. If unspecified, defaults to the telephony platform's internal setting.

4.3 Object methods:

4.3.1 Start

Enable DTMF interruption and start a DTMF reading session.

Syntax:

Object.Start( );

Return value:

None

Exception:

None

4.3.2 Stop

Disable DTMF. The key strokes entered by the user, however, remain in the buffer.

Syntax:

Object.Stop( );

Return value:

None

Exception:

None

4.3.3 Flush

Flush the DTMF buffer. Flush can not be called during a DTMF session..

Syntax:

Object.Flush( );

Return value:

None

Exception:

None

4.4 Events

4.4.1 onkeypress

Fires when a DTMF key is press. This overrides the default event inherited from the HTML control. When user hits the escape key, the onRec event fires, not onKeypress.

Syntax:

Inline HTML	<DTMF onkeypress="*handler*" …>
Event property	Object.onkeypress = *handler* Object.onkeypress = GetRef("*handler*");

Event Object Info:

Bubbles	No
To invoke	Press on the touch-tone telephone key pad
Default action	Returns the key being pressed

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

4.4.2 onReco

Fires when a DTMF session is ended. The event disables the current DTMF object automatically.

Syntax:

Inline HTML	<DTMF onReco="*handler*" …>
Event property	Object.onReco = *handler* Object.onReco = GetRef("*handler*");

Event Object Info:

Bubbles	No
To invoke	User presses the escape key or the number of key strokes meets specified value.
Default action	Returns the key being pressed

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

4.4.3 onTimeout

Fires when no phrase finish event is received before time out. The event halts the recognition process automatically.

Syntax:

Inline HTML	<DTMF onTimeout="*handler*" …>
Event property (in ECMAScript)	Object.onTimeout = *handler* Object.onTimeout = GetRef("*handler*");

Event Object Info:

Bubbles	No
To invoke	No DTMF key stroke is detected within the timeout specified.
Default action	None

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

5 CallControl Object

Represents the telephone interface (call, terminal, and connection) of the telephone voice browser. This object is as native as window object in a GUI browser. As such, the lifetime of the telephone object is the same as the browser instance itself. A voice browser for telephony instantiates the telephone object, one for each call. Users don't instantiate or dispose the object.

At this point, only features related to first-party call controls are exposed through this object.

5.1 Properties

- **address**
  Read-only. XML DOM node object. Implementation specific. This is the address of the caller. For PSTN, may a combination of ANI and ALI. For VoIP, this is the caller's IP address.

- **ringsBeforeAnswer**
  Number of rings before answering an incoming call. Default is infinite, meaning the developer must specifically use the Answer( ) method below to answer the phone call. When the call center uses ACD to queue up the incoming phone calls, this number can be set to 0.

5.2 Methods

Note: all the methods here are synchronous.

5.2.1 Transfer

Transfers the call. For a blind transfer, the system may terminate the original call and free system resources once the transfer completes.

Syntax:

telephone.Transfer(strText);

Parameters:

- **strText**: Required. The address of the intended receiver.

Return value:

None.

Exception:

Throws an exception when the call transfer fails. e.g., when end party is busy, no such number, fax or answering machine answers.

5.2.2 Bridge

Third party transfer. After the call is transferred, the browser may release resources allocated for the call. It is up to the application to recover the session state when the transferred call returns using strUID. The underlying telephony platform may route the returning call to a different browser. The call can return only when the recipient terminates the call.

Syntax:

telephone.Bridge(strText, strUID, [imaxTime]);

Parameters:

- **strText**: Required. The address of the intended receiver.
- **strUID**: Required. The session ID uniquely identifying the current call. When the transferred call is routed back, the srtUID will appear in the address attribute.
- **imaxTime**: Optional. Maximum duration in seconds of the transferred call. If unspecified, defaults to platform-internal value

Return value:

None.

Exception:

None.

5.2.3 Answer

Answers the phone call.

Syntax:

telephone.Answer( );

Return value:

None.

Exception:

Throws an exception when there is no connection. No onAnswer event will be fired in this case.

5.2.4 Hangup

Terminates the phone call. Has no effect if no call currently in progress.

Syntax:

telephone.Hangup( );

Return value:

None.

Exception:

None.

5.2.5 Connect

Starts a first-party outbound phone call.

Syntax:

telephone.Connect(strText, [iTimeout]);

Parameters:

- **strText**: Required. The address of the intended receiver.
- **iTimeout**: Optional. The time in milliseconds before abandoning the attempt. If unspecified, defaults to platform-internal value.

Return value:

None.

Exception:

Throws an exception when the call cannot be completed, including encountering busy signals or reaching a FAX or answering machine (Note: hardware may not support this feature).

5.2.6 Record

Record user audio to file.

Syntax:

telephone.Record(url, endSilence, [maxTimeout], [initialTimeout]);

Parameters:

- **url**: Required. The url of the recorded results.
- **endSilence**: Required. Time in milliseconds to stop recording after silence is detected.
- **maxTimeout**: Optional. The maximum time in seconds for the recording. Default is platform-specific.
- **initialTimeout**: Optional. Maximum time (in milliseconds) of silence allowed at the beginning of a recording.

Return value:

None.

Exception:

Throws an exception when the recording can not be written to the url.

5.3 Event Handlers

App developers using telephone voice browser may implement the following event handlers.

5.3.1 onIncoming( )

Called when the voice browser receives an incoming phone call. All developers can use this handler to read caller's address and invoke customized features before answering the phone call.

5.3.2 onAnswer( )

Called when the voice browser answers an incoming phone call.

5.3.3 onHangup( )

Called when user hangs up the phone. This event is NOT automatically fired when the program calls the Hangup or Transfer methods.

5.4 Example

This example shows scripting wired to the call control events to manipulate the telephony session.

```
<HTML>
<HEAD>
  <TITLE>Logon Page</TITLE>
  </HEAD>
    <SCRIPT>
     var focus;
     function RunSpeech() {
          if (logon.user.value == "") {
                focus="user";
                p_uid.Start(); g_login.Start(); dtmf.Start();
     return;
          }
          if (logon.pass.value == "") {
                focus="pin";
```



```
                p_pin.Start(); g_login.Start(); dtmf.Start();
     return;
          }
          p_thank.Start(); logon.submit();
     }
     function login_reco() {
          res = event.srcElement.recoResult;
          pNode = res.selectSingleNode("//uid");
          if (pNode != null)
                logon.user.value = pNode.xml;
          pNode = res.selectSingleNode("//password");
          if (pNode != null)
                logon.pass.value = pNode.xml;
     }
     function dtmf_reco() {
          res = event.srcElement.dtmfResult;
          if (focus == "user")
                logon.user.value = res;
          else
                logon.pin.value = res;
     }
    </SCRIPT>
  <SCRIPT for="callControl" event="onIncoming">
     <!--
          // read address, prepare customized stuff if any
          callControl.Answer();
     //
    </SCRIPT>
  <SCRIPT for="callControl" event="onOffhook">
     <!--
          p_main.Start(); g_login.Start(); dtmf.Start();
     focus="user";
     //
   </SCRIPT>
  <SCRIPT for="window" event="onload">
     <!--
          if (logon.user.value != "") {
                p_retry.Start();
                logon.user.value = "";
                logon.pass.value = "";
                checkFields();
          }
     //
    </SCRIPT>
<BODY>
  <reco id="g_login"
        onReco="login_reco(); runSpeech()"
```

```
        timeout="5000"
        onTimeout="p_miss.Start(); RunSpeech()" >
        <grammar
    src=http://kokaneel/etradedemo/speechonly/login.xml/>
        </ reco >
  <dtmf id="dtmf"
        escape-"#"
        onkeypress="g_login.Stop();"
        onReco="dtmf_reco();RunSpeech()"
        interdigitTimeout="5000"
        onTimeout="dtmf.Flush(); p_miss.Start();RunSpeech()" />

  <prompt id="p_main">Please say your user I D and pin
   number</prompt>
  <prompt id="p_uid">Please just say your user I D</prompt>
  <prompt id="p_pin">Please just say your pin number</prompt>
  <prompt id="p_miss">Sorry, I missed that</prompt>
  <prompt id="p_thank">Thank you. Please wait while I verify
   your identity</prompt>
  <prompt id="p_retry">Sorry, your user I D and pin number do
   not match</prompt>

  <H2>Login</H2>
<form id="logon">
        UID:   <input name="user" type="text"
        onChange="runSpeech()" />
        PIN:   <input name="pass" type="password"
        onChange="RunSpeech()" />
 </form>
 </BODY>
 </HTML>
```

6 Controlling dialog flow

6.1 Using HTML and script to implement dialog flow

This example shows how to implement a simple dialog flow which seeks values for input boxes and offers context-sensitive help for the input. It uses the title attribute on the HTML input mechanisms (used in a visual browser as a "tooltip" mechanism) to help form the content of the help prompt.


```
<html>
  <title>Context Sensitive Help</title>
<head>
  <script>   var focus;
        function RunSpeech() {
             if (trade.stock.value == "") {
                  focus="trade.stock";
                  p_stock.Start();
                  return;
             }
             if (trade.op.value == "") {
                  focus="trade.op";
                  p_op.Start();
                  return;
             }
             //.. repeat above for all fields
             trade.submit();
        }
        function handle() {
             res = event.srcElement.recoResult;
             if (res.text == "help") {
                  text = "Please just say";
                  text += document.all[focus].title;
                  p_help.Start(text);
             } else {
                  // proceed with value assignments
                  }
             }
   </script>
   </head>
<body>
   <prompt id="p_help" onComplete="checkFileds()" />
   <prompt id="p_stock" onComplete="g_stock.Start()">Please say
   the stock name</prompt>
   <prompt id="p_op" onComplete="g_op.Start()">Do you want to
   buy or sell</prompt>
   <prompt id="p_quantity" onComplete="g_quantity.Start()">How
   many shares?</prompt>
   <prompt id="p_price" onComplete="g_price.Start()">What's the
   price</prompt>
```

```
<reco id="g_stock" onReco="handle(); checkFields()" >
     <grammar src="./g_stock.xml" />
</ reco >

<reco id="g_op" onReco="handle(); checkFields()" />
     <grammar src="./g_op.xml" />
</ reco >

<reco id="g_quantity" onReco="handle(); checkFields()" />
     <grammar src="./g_quant.xml" />
</ reco >

<reco id="g_price" onReco="handle(); checkFields()" />
     <grammar src="./g_quant.xml" />
</ reco >


<form id="trade">
     <input name="stock" title="stock name" />
     <select name="op" title="buy or sell">
          <option value="buy" />
          <option value="sell" />
          </select>
     <input name="quantity" title="number of shares" />
     <input name="price" title="price" />
</form>
</body>
</html>
```

6.2 Using SMIL

The following example shows activation of prompt and reco elements using SMIL mechanisms.

```
<html xmlns:t="urn:schemas-microsoft-com:time"
      xmlns:sp="urn:schemas-microsoft-com:speech">
<head>
<style>
  .time { behavior: url(#default#time2); }
</style>
</head>
<body>

<input name="txtBoxOrigin" type="text"/>
<input name="txtBoxDest" type="text" />
```



```
<sp:prompt class="time" t:begin="0">
    Please say the origin and destination cities
</sp:prompt>

<t:par t:begin="time.end" t:repeatCount="indefinitely"
  <sp:reco class="time"  >
    <grammar src="./city.xml" />
    <bind targetElement="txtBoxOrigin"
     value="//origin_city" />
    <bind targetElement="txtBoxDest"
     test="/sml/dest_city[@confidence $gt$ 40]"
     value="//dest_city" />
  </sp:reco>
</t:par>

</body>
</html>
```

7 SMEX (Message) element/object

SMEX, short for Simple Messaging EXchange/EXtension, is an object that communicates with an external component or application on the platform of the client device. It can be embedded into an XML or similar markup based document as an element with the tag name <smex>. Exemplary usages of the messaging object may include logging and telephony controls. The object represents the extensibility of markup based recognition and prompting as it allows new functionality be added through messaging.

Upon instantiation, the object is directed to establish an asynchronous message exchange channel with a platform component or application through its configuration parameters or attribute specifications. It has a string property whose content is sent to the platform component or application whenever the property is the recipient of an

assignment operation (i.e., lvalue). Similarly, it also has a property of XML DOM Node type that holds the message received from the platform component or application. The message object sends an event whenever it receives a platform message. Since its basic operations are asynchronous, the object also has a built-in clock for the application developers to manipulate timeout settings.

The message or smex object is agnostic to the means of communications. However, in one embodiment, the smex object has the same life span as ordinary XML or markup elements, namely, the smex object will be destroyed when its hosting document is unloaded. While in many cases, the smex object can perform automatic clean-up and release communication resources when it is unloaded, there might be use cases (e.g., call controls) in which a persistent communication link is desirable across markup pages. For those cases, the architecture places the responsibility of relinquishing the allocated resources (e.g. close the socket) on the application developers.

The smex object is neutral on the format (schema) of messages. In some embodiments, it may desirable to require implementers to support a few rudimentary schemas, with strong preferences to existing standard messages formats (e.g. as used in SIP or CCXML). In essence, the architecture allows both the platform and application developers to take the full advantage of the standardized extensibility of XML or similar markup to introduce other features without losing interoperability in the mean time.

Example 1: use of smex as a logging object


```
<smex id="logServer">
     <param name="d:server" xmlns:d="urn:Microsoft.com/COM">
          <d:protocol>DCOM</d:protocol>
          <d:clsid>209309302930202932094209843209B</d:clsid>
          <d:iid>0903859304903498530985309094803</d:iid>
     </param>
</smex>

<listen …>
     …// other directives binding reco results to input
fields
     <bind targetElement="logServer" targetAttribute="sent"
          value="*[@log $ge$ 3]"/>
</listen>
```

This example demonstrates how a logging mechanism can be achieved using a COM object with its class id and interface id. The speech developers attach an attribute "log" indicating the level of interests for logging to the relevant SML nodes. In the example above, the app developer chooses to log all nodes with log value greater or equal to 3 by using a single bind directive. The example works in both downlevel and uplevel browsers.

The example also intends to demonstrate it is possible for a page to contain multiple smex objects communicated with the same platform component as long as there won't be confusion on which smex object is responsible for delivering the platform messages back to the recognition document. The above example implies a component can implement multiple interfaces, each of which has its own smex or message

conduit. The same argument applies to TCP servers listening to multiple ports.

Example 2: Reading the addresses for an incoming call:

```
<input type="text" id="remote"/>
<input type="text" id="transfer"/>
<input type="text" id="local"/>
<input type="hidden" id="session_id"/>
…
<smex id="telephone" sent="start_listening">
     <param name="server">http://tel-svr/whatever</param>
     <bind targetElement="session_id" value="//sid"/>
     <bind targetElement="remote" value="//remote_addr"/>
     <bind targetElement="transfer"
value="//transfer_addr"/>
     <bind targetElement="local" value="//local_addr"/>
     …
</smex>
```

This example shows how the bind directives can be used to process the received message. The example assumes a message for incoming call to have sub-elements remote_addr, transfer_addr, and local_addr whose contents represent the, remote, transfer, and local addresses of the incoming call, respectively.

In this example, a connectionless protocol based on HTTP is used to communicate with the telephony server. The telephony server here is designed to communicate with more than one browser instances, and hence each client must identify itself with a unique id assigned by the server when the app

starts. This is achieved in this example by sending the server a "start_listening" message. In this example, the session id is stored in a hidden field that can be sent back to the Web server and passed on to the next page of the application, although other techniques (e.g. client side cookie) can also be used to manage session state. As is in the case for reco, not every bind directives will be executed for every platform message. The above example does not imply the unique id will only be received when there is an incoming phone call.

7.1 *Properties*

The smex object can have the following properties, with only the Read/Write properties allowed to also serve as attributes for the initial value specification.

- **sent**: Read/Write, a string corresponding to the message to be sent to the platform component. Whenever sent is used as a lvalue, its contents are dispatched. There is no effect when the property is used as a rvalue or when a null object is assigned to this property.
- **received**: Read Only, an XML DOM Node data indicating the received message. The message will be available as a rvalue until the next **onReceive** event is ready to send.
- **timer**: Read/Write, a number in millisecond indicating the time span before a timeout event will be triggered. The clock starts ticking when the property is assigned a positive value. The value can be changed when a count down is in progress. A zero or negative value stops the clock without triggering the timeout event. The default is 0, meaning no timeout.

- **status**: Read Only, an integer indicating the recent status of the object. The possible values are 0, -1, and -2, which means normal, timeout expired, and communication with the platform cannot be established or has been interrupted, respectively. Platform specific error messages should conveyed through the received property. For the cases that the error message is successfully delivered, the status code is 0.

7.2 *Events*

The object has the following events:

- **onReceive**: This event is sent when a platform message has arrived. If there are any directives declared by the **bind** elements, those directives will first be evaluated before the event is fired. Prior to the sending, the **received** property will be updated.
- **onError**: This event is sent when the timeout expires, or a communication link error has been encountered. When the event is sent, the **status** property will be updated with a corresponding error code as described above.

7.3 *Child Elements*

When assuming an element form, smex may have the following child elements:

- **bind**: same as in reco, except the directives are operated on the received message.
- **param**: same as in reco, provides platform specific parameters for the smex object. Each param element may be named using a "**name**" attribute, with the contents of the param element being the value of the parameter. In

one embodiment, the element should understand standard XML attributes for name space and XML data type declaration.

7.4 *Other Comments*

One elegant way to extend SMEX for the logging function would be

```
<smex id="logServer" …> … </smex>
<script> function logMessage(logClass, message) {
     logServer.sent = logClass + "|" + message;
} </script>
```

This in effect extends the object with a (global) function whose behavior can be customized. In the above example, the logging function is programmed to insert a field separator "|" between the id and the message.

For those who do not like global functions, they can use the "prototype" property of ECMAScript to attach the function as an object method. For example

```
<smex id="logServer" onload="addFunction()"> … </smex>
<script>
     function my_logMessage(logClass, message) {
          logServer.sent = logClass + "|" + message;
     }
     function addFunction() {
```

```
        logServer.prototype.logMessage =
my_logMessage;
    }
</script>
```

One can refer to the function in a more objected-oriented manner:

```
    logServer.logMessage(RECO_LOG_ERROR, "My
message");
```

It is noted that more work is required from the smex object implementers in order to make the extension work as in the above example, although all the necessary mechanisms are already well established standards.

4. Brief Description of the Drawings

FIG. 1 is a plan view of a first embodiment of a computing device operating environment.

FIG. 2 is a block diagram of the computing device of FIG. 1.

FIG. 3 is a plan view of a telephone.

FIG. 4 is a block diagram of a general purpose computer.

FIG. 5 is a block diagram of an architecture for a client/server system.

FIG. 6 is a display for obtaining credit card information.

FIG. 7 is a page of mark-up language executable on a client.

FIG. 8 is an exemplary page of mark-up language executable on a client having a display and voice recognition capabilities.

FIGS. 9A and 9B are an exemplary page of mark-up language executable on a client with audible rendering only and system initiative.

FIG. 10A and 10B are an exemplary page of mark-up language executable on a client with audible rendering only and mixed initiative.

FIG. 11 is an exemplary script executable by a server side plug-in module.

FIG. 12 is a pictorial illustration of a first operational mode of a recognition server.

FIG. 13 is a pictorial illustration of a second operational mode of the recognition server.

FIG. 14 is a pictorial illustration of a third operational mode of the recognition server.

FIGS. 15A and 15B are an exemplary page of declarative mark-up language executable on a client without scripting.

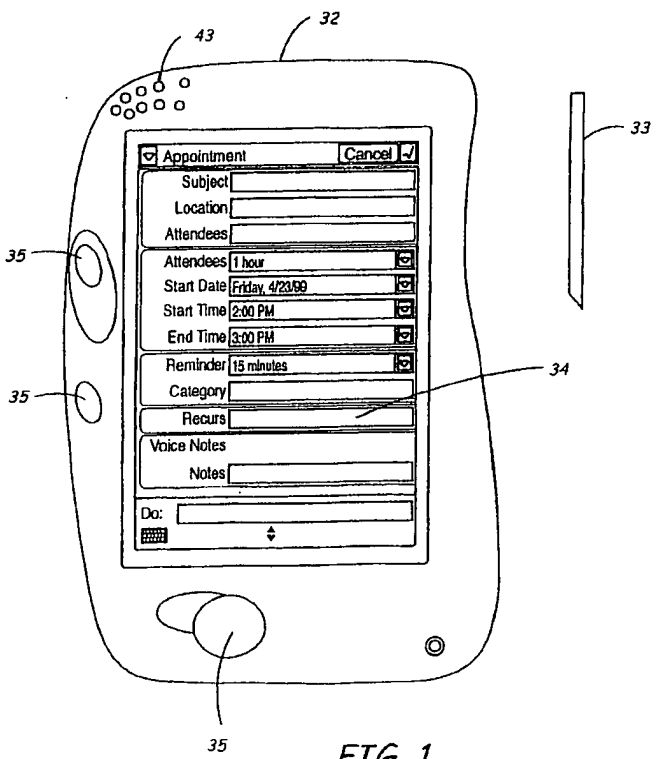

43
32
33
35
35
34
35
Appointment
Cancel
Subject
Location
Attendees
Attendees 1 hour
Start Date Friday, 4/23/99
Start Time 2:00 PM
End Time 3:00 PM
Reminder 15 minutes
Category
Recurs
Voice Notes
Notes
Do:

FIG. 1

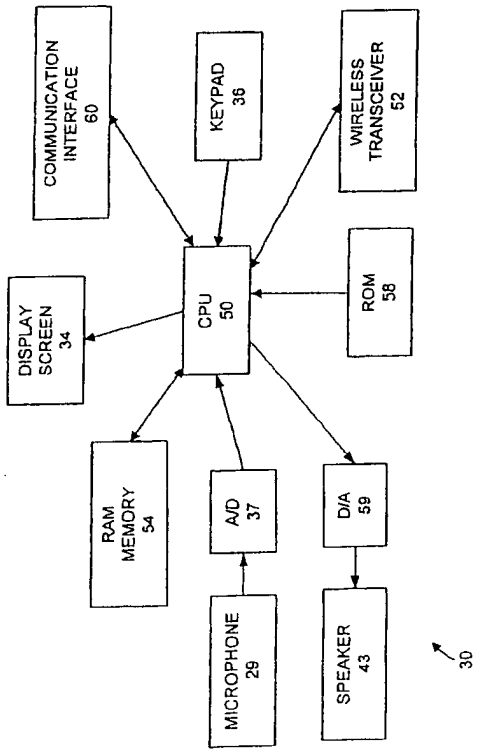
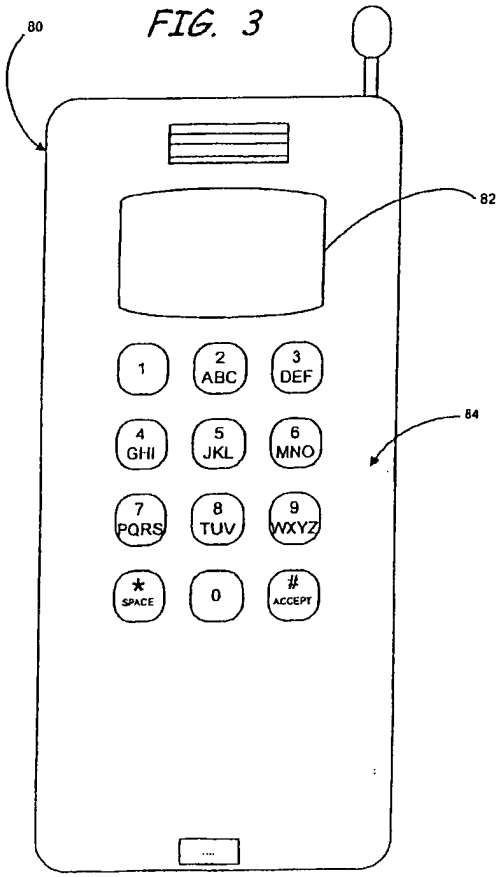

COMMUNICATION INTERFACE 60
KEYPAD 36
WIRELESS TRANSCEIVER 52
DISPLAY SCREEN 34
CPU 50
ROM 58
RAM MEMORY 54
A/D 37
D/A 59
MICROPHONE 29
SPEAKER 43
30

FIG.2

FIG. 3


80
82
84
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
* SPACE
0
ACCEPT

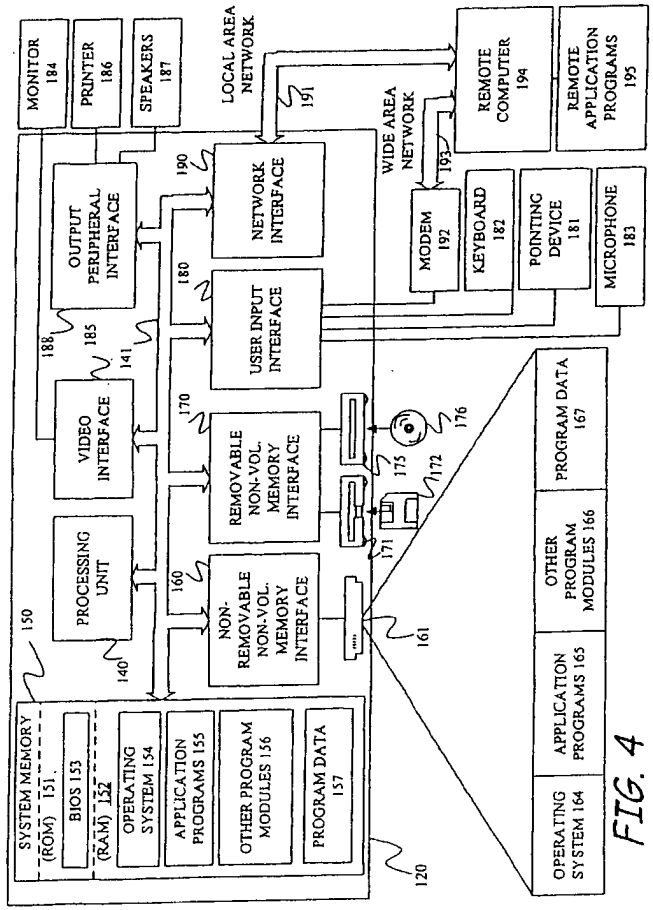

MONITOR 184
PRINTER 186
SPEAKERS 187
LOCAL AREA NETWORK
191
WIDE AREA NETWORK
193
REMOTE COMPUTER 194
REMOTE APPLICATION PROGRAMS 195
OUTPUT PERIPHERAL INTERFACE
NETWORK INTERFACE
MODEM 192
KEYBOARD 182
POINTING DEVICE 181
MICROPHONE 183
USER INPUT INTERFACE
VIDEO INTERFACE
REMOVABLE NON-VOL. MEMORY INTERFACE
NON-REMOVABLE NON-VOL. MEMORY INTERFACE
PROCESSING UNIT
SYSTEM MEMORY (ROM) 151
BIOS 153
(RAM) 152
OPERATING SYSTEM 154
APPLICATION PROGRAMS 155
OTHER PROGRAM MODULES 156
PROGRAM DATA 157
PROGRAM DATA 167
OTHER PROGRAM MODULES 166
APPLICATION PROGRAMS 165
OPERATING SYSTEM 164

FIG. 4

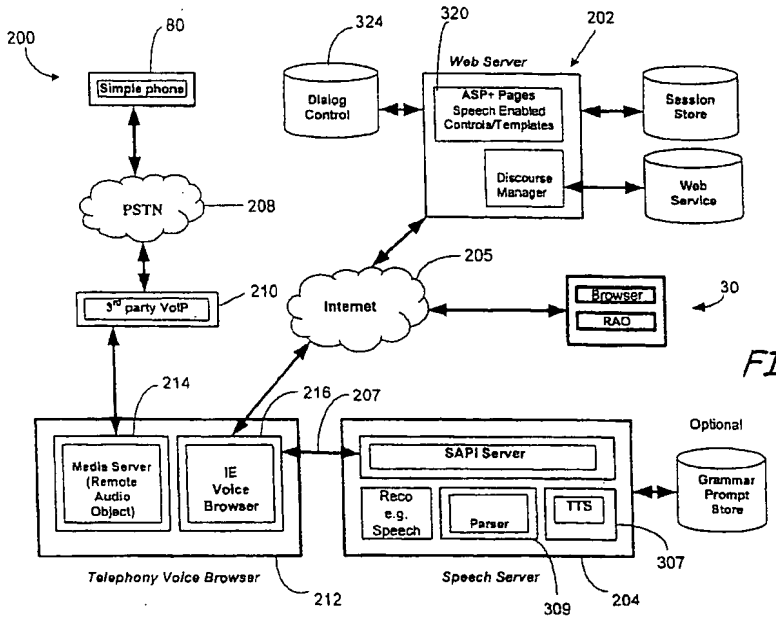


FIG. 5

(154) 特開2003-131772

```
<html>
  <form id="get_card_info" method="post" action="http://payment.asp">
    <select name="card_type">
      <option value="amex">American Express</option>
      <option value="visa">Visa</option>
      <option value="ms">MasterCard</option>
    </select>
    <input type="text" name="card_num" width="30"
      onChange="handle()" />
    <input type="text" name="expiry_date" />
    <input type="submit" value="Submit" onClick="verify()" />
  </form>
  <script> <![CDATA[
        function handle() {
          if (get_card_info.card_type.value == "amex") {
              if (get_card_info.card_num.length != 15)
                  alert ("amex should have 15 digits");
          } else
              if (get_card_info.card_num.length != 16)
                  alert ("visa and master should have 16 digits");
        }
        function verify() {
          var flag = window.confirm("submit the credit card info?");
          if (flag)
              get_card_info.submit();
        }
    ]]>
  </script>
</html>
```

260

262

FIG. 7

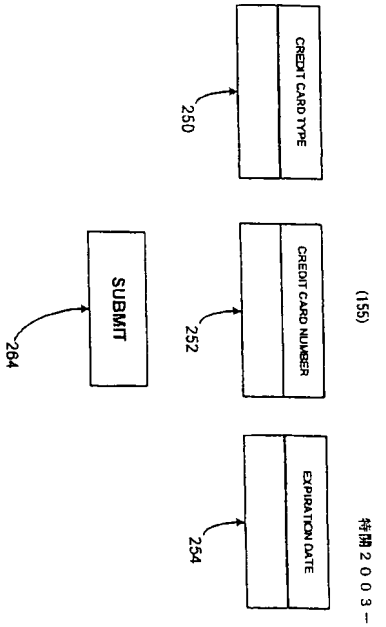


FIG. 6

(155) 特開2003-131772

```
270  <html>
       <form id="get_card_info" method="post" action="http://payment.asp">
        <select name="card_type" onClick="talk(g_card_types)">
          <option value="amex">American Express</option>
          <option value="visa">Visa</option>
          <option value="ms">MasterCard</option>
        </select>
        <reco id="g_card_types" onReco ="handle()" >
290       <grammar  src="./gram#card_types" />
        </reco >
        <input type="text" name="card_num" width="30"
          onClick="talk(g_card_num)" />
        <reco id="g_card_num" onReco ="handle()" >
          <grammar  src="./gram#digits" />
        </reco >
        <input type="text" name="expiry_date"
          onClick="talk(g_expiry_date)" />
        <reco id="g_expiry_date" >
          <grammar  src ="  ./gram#dates" />
        </reco >
        <input type="submit" value="Submit" onClick="verify()" />
       </form>
       <script><![CDATA[
              function talk(gobj) {
                  gobj.activate();
              }
              function handle() {
                  if (get_card_info.card_num != null)  {
                      if (get_card_info.card_type.value == "amex") {
                          if (get_card_info.card_num.length != 15)
                              alert ("amex should have 15 digits");
                      } else
                          if (get_card_info.card_num.length != 16)
                              alert ("visa and master should have 16 digits");
                  }
              }
              function verify() {
                  var flag = window.confirm("submit the credit card
           info?");
                  if (flag) {
                      get_card_info.submit();
                  }
              }
          ]]>
       </script>
     </html>
```

280
282
284
272

FIG. 8


```
<body>
  <form id=" get_card_info " method =" post" action =" http://payment.asp "
    onActivate =" welcom() ">
  <prompt id =" p_welcome "> We now need your credit card </prompt>
  <prompt id =" p_mumble "> I didn't understand you </prompt>
  <prompt id =" p_card_type " bargein ="true"> What credit card would you
300 use? </prompt>
  <prompt id =" p_card_num " bargein ="true"> Please say the number" </prompt>
  <prompt id =" p_expiry_date " bargein ="true"> What is the expiration
date? </prompt>
  <prompt id =" p_content ">
303       I have your  <value select =" card_type " /> <value select =" card_num " />
          with expiration date
        <value select =" expiry_date " />
  </prompt>
  <prompt id =" p_confirm "> Is this correct? </prompt>
  <reco id =" g_card_types "     onNoReco =" mumble(this, 2) ">
   onReco =" _handle(this, card_type) " />
  <grammar src =" ./gram#card_types "
  </reco >
  <reco id =" g_card_num " onNoReco =" mumble(this, 1) ">
   onReco =" _handle(this,  card_num) " />
   <grammar src =" ./gram#digits " />
  </reco >
305 <reco id =" g_expiry_date " onNoReco =" mumble(this, 1) ">
   onReco =" _handle(this, expiry_date) " audio =" rec " />
    <grammar src =" ./gram#dates " />
  </reco >
  <reco id =" confirmation "    onReco     =" confirmed(this) " />
    <grammar src =" ./gram#yesno " />
  </reco >
  <select name =" card_type ">
     <option value =" amex "> American Express </option >
     <option value =" visa "> Visa </option >
     <option value =" ms "> MasterCard </option >
   </select >
   <input type =" text" name =" card_num " width =" 30 " />
   <input type =" text" name =" expiry_ date " />
   <input type =" submit " value =" Submit " />
</form >
<script ><![CDATA[
       function welcome() {
         p_welcome.active();
         repeat = 0;
         checkfilled();
       }
       function mumble(gobj, maxprompts) {
302      gobj.deactivate();
         p_mumble.active();
         checkfilled();
       }
```

TO FIG. 9B

FIG. 9A

```
    function _handle() {
     handle();
     checkFilled();
    }
    function checkFilled() {
      if (card_type.value == "") {
            p_card_type.active();
            g_card_types.activate(); return;
      }
      if (card_num.value =="") {
            p_card_num.active(); g_card_num.acti   vate(); return;
      }
      if (expiry_date.value == "") {
            p_expiry_data.active(); g_expiry_date.activate(); return;
      }
        p_content.activate();
       p_confirm.activate();
       confirmation.activate();
    }
      function confirmed(gobj) {
            if (gobj.recogRes.text == "yes")
                  get_card_info.submit();
      }
      //
      // user codes start here
      //
      function handle() {

        if (get_card_info.card_type == "amex") {
            if (get_card_info.card_num.length != 15) {
                prompt.speak ("amex should have 15 digits");
                get_card_info.card_num = "";
            }
        } else
            if (get_card_info.card_num.length != 16)  {
                prompt.speak ("visa and master should have 16 digits");
                get_card_info.card_num = "";
            }
      }
    ]]>
  </script>
</body>
```

FIG. 9B


```
<body>
    <form id="get_card_info" method="post" action="http://payment.asp"
      onactivate="welcome()">
      <prompt id="p_welcome">We now need your credit card</prompt>
      <prompt id="p_mumble">I didn't understand you</prompt>
      <prompt id="p_card_type" bargein="true">What credit card would you
use?</prompt>
      <prompt id="p_card_num" bargein="true">Please say the number</prompt>
      <prompt id="p_expiry_date" bargein="true">What is the expiration
date?</prompt>
      <prompt id="p_content">
        I have your  <value select="card_type" /><value select="card_num" />
  with expiration date <value select="expiry_date" />
      </prompt>
      <prompt id="p_confirm">Is this correct?</prompt>
      <reco id="g_get_card_info" onReco="_handle()"  onNoReco ="mumble(this)">
        <grammar   src  ="./gram/getPayment" />
        <bind target="card_type" value="/card/type" />
        <bind target="card_num" value="/card/number" />
405 ——  <bind target="expiry_date" value="/card/expr_date" />
      </reco>
      <reco id="confirmation" onReco ="confirmed(this)  onNoReco="mumble(this)" />
        <grammar src ="./gram#yesno"/>
      </reco >
      <select name="card_type">
        <option value="amex">American Express</option>
        <option value="visa">Visa</option>
        <option value="ms">MasterCard</option>
      </select>
      <input type="text" name="card_num" width="30" />
      <input type="text" name="expiry_date" />
      <input type="submit" value="Submit" />
    </form>
    <script><![CDATA[
        function welcome() {
          p_welcome.active();
          repeat = 0;
          checkFilled();
        }
        function mumble(gobj) {
          gobj.deactivate();
          p_mumble.active();
          checkFilled();
        }
        function _handle() {
          handle();
          checkFilled();
        }
```

TO FIG. 10B

FIG. 10A

```
        function checkFilled()   {
          if (card_type.value == "") {
                p_card_type.acrive();; do_field.activate(); return;
          }
          if (card_num.value =="") {
                p_card_num.active(); do_field.activate(); return;
          }
          if (expiry_date.value == "") {
                p_expiry_data.active(); do_field.activate(); return;
          }
          p_content.activate();
          p_confirm.activate();
          confirmation.activate();
        }
        function confirmed(gobj) {
          if (gobj.recogRes.text == "yes")
                get_card_info.submit(gensml());
        }
        //
        // user codes start here
        //
          function handle() {
          if (field == get_card_info.card_num) {
              if (get_card_info.card_num.length != 15) {
                  prompt.speak ("amex should have 15 digits");
                  get_card_info.card_num = "";
              }
          } else
              if (get_card_info.card_num.length != 16)  {
                  prompt.speak ("visa should have 15 digits");
                  get_card_info.card_num = "";
              }
          }
            function gensml() {
                  str = '<sml><credit_card type="';
                  str += card_type.value; str += '"><number>';
                  str += card_number.value; str += '</number><expire>';
                  str += expiry_date.value; str +=
'</expire></credit_card></sml>';
                  return str;
            }
        ]]>
      </script>
    </body>
```

FIG. 10B

SAMPLE ASP+ PAGE

```
<%@ Page language="Jscript" AutoEventWireup="false" Inherits="Credit.Transaction" %>
<html><head>
<!-- ASPX page for both voice-only & multimodal credit card example -->
<script>
function handle() {
        if (field == get_card_info.card_num) {
           if (get_card_info.card_num.length != 15) {
             prompt.speak ("amex should have 15 digits");
             get_card_info.card_num = "";
           }
        } else
           if (get_card_info.card_num.length != 16)  {
             prompt.speak ("visa should have 15 digits");
             get_card_info.card_num = "";
           }
      }
function gensml() {
      str = '<sml><credit_card><card_type>';
      str += card_type.value; str += '</card_type><number>';
      str += card_number.value; str += '</number><expire>';
      str += expiry_date.value; str += '</expire></credit_card></sml>';
      return str;
}
</script>
<script runat="server">
function Page_Load (obj, args) {
      if (PostBack) {
        validator = new System.Speech.SMLValidator("./CreditSDL.xml");
        dsml = validator.Evaluate(args);
        Navigate(ChoosePage(dsml));
      } else {
        // initialize fields with args
      }
}
</script>
</head>
<body>
<speech:form id="get_card_info" style="system_initiative"
      prompt="./prompt/getPayment" onsubmit="gensml()">
  <speech:choice name="card_type" prompt="What credit card would you use?"
      grammar="./gram/card_types" onPhraseFinish="handle()"
      <option>American Express</option>
      <option>Visa</option>
      <option>Mastercard </option>
  </speech:choice>
  <speech:textbox name="card_number" prompt="Please say the number"
      grammar="./gram/digits" onPhraseFinish="handle()">
  <speech:textbox name="expiry_date" prompt="What is the expiration date? "
      grammar="./gram/dates" onPhraseFinish="handle()"/>

</speech:form>
</body>
</html>
```

FIG. 11

FIG. 12

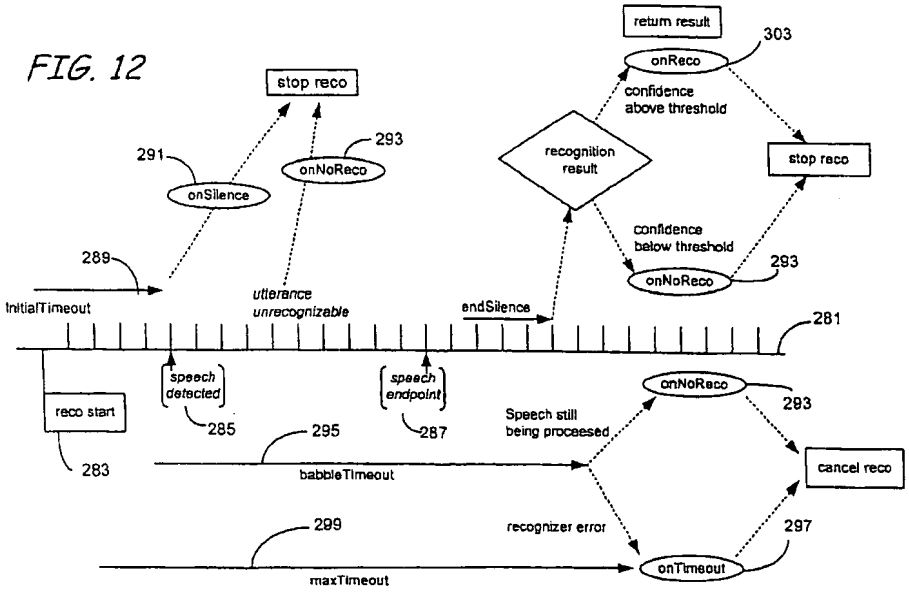
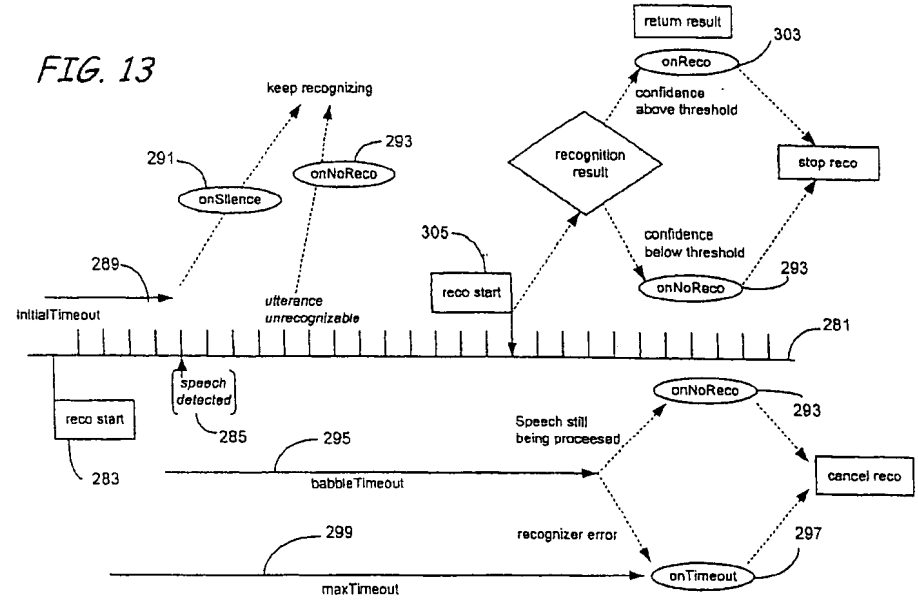

return result
303
onReco
confidence above threshold
stop reco
recognition result
confidence below threshold
onNoReco
293
291
onSilence
289
InitialTimeout
utterance unrecognizable
endSilence
281
speech detected
285
speech endpoint
287
reco start
283
295
babbleTimeout
Speech still being proceesed
cancel reco
299
recognizer error
297
onTimeout
maxTimeout

FIG. 13


return result
303
onReco
keep recognizing
confidance above threshold
stop reco
recognition result
291
293
onNoReco
onSilence
305
confidence below threshold
289
reco start
InitialTimeout
utterance unrecognizable
281
speech detected
285
onNoReco
293
reco start
295
Speech still being proceesed
cancel reco
283
babbleTimeout
299
recognizer error
297
onTimeout
maxTimeout

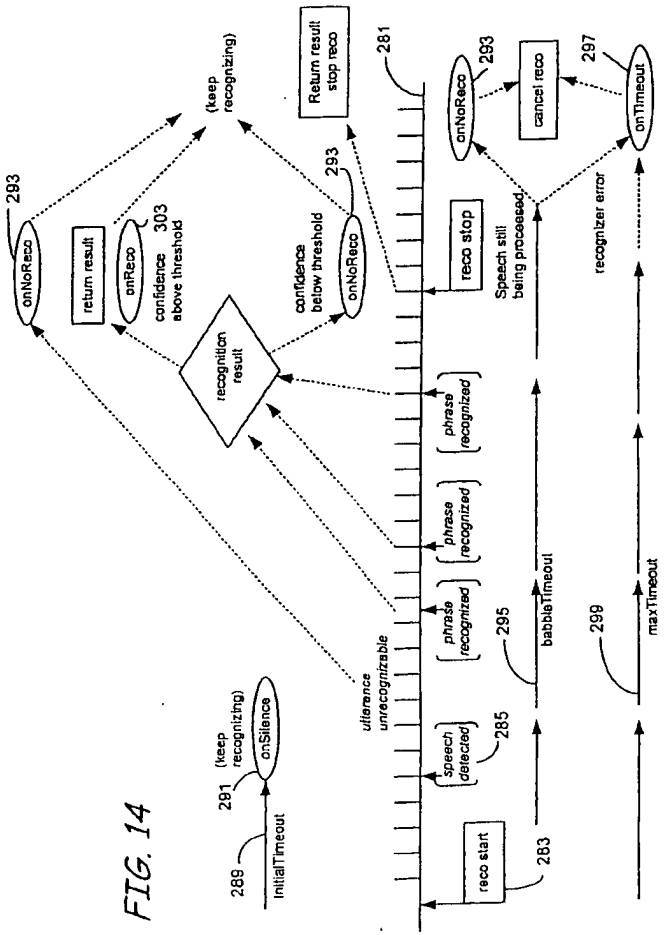


FIG. 14


```
<html>
<body>
<!- the data section -->
  <form id="get drink">
    <input name="drink" />
    <input type="radio" name="cream"/>
    <input type="radio" name="sugar"/>
    <input name="uid" type="hidden"/>
  </form>
<!-- The speech section -->
<prompt id="welcome">Welcome, caller! </prompt>
<prompt id="ask"> Do you want coke, coffee, or orange juice?
</prompt>
<prompt id="confirm"> I heard <value href="drink"/>. Is this
     correct? </prompt>
<prompt id="thanks"> Thank you. Please wait when I get it for
     you. </prompt>
<prompt id="retry"> Okay, let's do this again </prompt>
<prompt id="reprompt"> Sorry, I missed that. </prompt>
<prompt id="cream_sugar">Do you want cream or sugar with your
     coffee?</prompt>

<reco id ="reco_drink"><grammar src="./drinktypes"/>
  <bind test="/[@confidence $le$ 10]"
    targetElement="reprompt" targetMethod="start"
    targetElement="ask" targetMethod="start"
    targetElement="reco_drink" targetMethod="start"/>
  <bind test="/drink/coffee[@confidence $gt$ 10]"
    targetElement="drink" value="/drink"
    targetElement="cream_sugar" targetMethod="start"
    targetElement="reco_cream_sugar"
          targetMethod="start"/>
  <bind test="/[@confidence $gt$ 10]"
    targetElement="drink" value="/drink"
    targetElement="confirm" targetMethod="start"
    targetElement="reco_yesno" targetMethod="start"/>
</reco>
```

FIG. 15A

```
<reco id="reco_cream_sugar"><grammar src="./cream+sugar"/>
     <bind test="/[@confidence $gt$ 10 and
          host()/get_drink/drink = 'coffee']"
          targetElement="cream" targetAttribute="checked"
               value="/cream/@value"
          targetElement="sugar" targetAttribute="checked"
               value="/sugar/@value"
          targetElement="confirm" targetMethod="start"
          targetElement="reco_yesno" targetMethod="start"/>
</reco>

<reco id="reco_yesno"> <grammar src="./yesno"/>
          <bind test="/yes[@confidence $gt$ 10]"
        384 —— targetElement="thanks" targetMethod="start"
        386 —— targetElement="get_drink"
                 targetMethod="submit" />
          <bind test="/no or ./[@confidence $le$ 10]"
        390 —— targetElement="retry" targetMethod="start"
        392 —— targetElement="ask" targetMethod="start"
        394 —— targetElement="reco_drink"
                 targetMethod="start"/>
</reco>

<!-call control section -->
<smex id="telephone" sent="start_listening"><param
server="ccxmlproc"> … </param>
          <bind targetElement="uid" value="/@uid"/>
          <bind test="/Call_connected"
     361 —— targetElement="welcome" targetMethod="start"
     362 —— targetElement="ask" targetMethod="start"
     363 —— targetElement="reco_drink"
                 targetMethod="start"/>
</smex>
</body>
</html>
```

FIG. 15B

1. Abstract

A markup language for execution on a client device in a client/server system includes extensions for recognition.

2. Representative Drawing

FIG. 1